TOWARDS AN AUTOMATED AND CUSTOMIZABLE LINEAR CRYPTANALYSIS OF A SUBSTITUTION-PERMUTATION NETWORK CIPHER

by

Bipeen Acharya

2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The increasing scale of the Internet of Things exposes the proprietors and users to a number of security threats in terms of invasion of privacy, identity thefts and the like. It is therefore absolutely necessary that the Internet of Things be secure so that communications between these computing devices can happen undisturbed by any illegitimate third party. Lightweight cryptography has emerged as one of the best solutions to make the communication secure. Cryptanalysis of a cipher is important to measure and evaluate the strength of a cipher. In this thesis, I explain in detail the process of linear cryptanalysis on a Substitution Permutation Network of 4 rounds. I provide an in-depth background of all key concepts and ideas to make this paper accessible to all kinds of readers. I also provide an open-source Python implementation of the linear cryptanalysis, which can be easily customized and expanded to perform stronger cryptanalysis or to extend to other cryptanalysis techniques. This paper provides a sound framework on linear cryptanalysis, which can be built upon for further research on cryptanalysis techniques.

Chapter 1
INTRODUCTION

1.1 The Internet of Things

The definition of the Internet of Things (IoT) is somewhat unclear and is subject to discussion. There has been a lot of commentary on how it should be defined, and there is not a common consensus yet. Variations in the definition also stem from geographical boundaries; while places like China and Europe have accepted the term Internet of Things, the United States refers to it with phrases like smart grid and smart device [1]. Regardless of what the definition is, there is a clear understanding of what the IoT generally entails. Van Kranenburg et al. [1] mentions that the IoT is driven by two major ideas: first, the increasing ubiquity of internet connectivity, and second, the drastic miniaturization of computing devices. Therefore, the Internet of Things is simply a large network of connected computing devices. While it is usually the case that devices interact with humans, the IoT contains and will contain devices that will interact with one another without necessitating human interaction. It is estimated that around 50 billion devices will be connected by the year 2020 [2].

The systems mentioned above encompass a large number of computing devices that are in use today. These devices are programmed to do very specific tasks. Microprocessors, Radio Frequency Identification (RFID) tags, sensor networks and mobile devices are examples of these systems that are becoming increasingly inter connected and ubiquitous in their use [3].

1.2 Motivation for Research

The IoT is growing, meaning, large and bulky computing devices are being replaced by small computing systems that still maintain the processing power and speed of the larger devices. Additionally, these systems are becoming increasingly connected to one another. This interconnection of thousands to millions of devices creates an intricate network of systems that interact with one another to provide a number of new applications for the devices. The applications can range from sensors in automobiles to protecting the environment by monitoring the air and water quality. Although the idea itself has been around for a while, it is only now becoming increasingly popular. With the development and advancement of new sensors and the decrease in price of computing technologies and power, the Internet of Things is becoming more integrated into everyday life.

The functionality provided by the Internet of Things comes at a cost. Being a part of the Internet of Things is the same as being a part of a large online database in that it poses a number of security threats to humans in the form of invasion of privacy, identity thefts and the like. A system that is a part of this network is subject to attacks from external sources. If it is compromised, it could result in serious consequences. For example, if someone attacks the system that a parent is using to monitor her baby, one can only imagine the consequences the situation could entail. Therefore, the advancement of the Internet of Things requires that the systems involved are secured from external attacks of any kind.

Achieving a satisfactory level of security for computing devices has been a challenge for the computing world. It is even more challenging to make the computing devices involved in the IoT secure because these devices are essentially very small computing devices with specific computational tasks. These devices do not have resources to allocate in providing robust security measures as most of their computational power is used in completing the task they are specifically programmed for. Security is never the prime objective of building an embedded system. Nevertheless, when a system is built, it so happens that security is imperative. Allocating resources to provide security could result in a decrease in efficiency of the device in terms of speed or in over consumption of power. Hence, these devices are limited as to what can be used to provide a satisfactory level of security.

Since the devices involved in the Internet of Things are connected to one another and must communicate frequently to keep the network operating to its maximum capability, there is no room for failure in these communications. To ensure that these communications are fail-proof, there should be no room for an eavesdropper to affect the communication. Even though most security approaches are undertaken only after the systems are well developed and in use, this cannot be the case here because of the structure of the Internet of Things and the amount of information and data at stake.

1.3 Possible solution

Lightweight cryptography, which entails ciphers specifically designed for devices with less computational power, seems to be one of the best approaches to achieve this goal. With encryption and decryption of messages between devices, it can be ensured that no intruder can play any role in disturbing communication between these systems. While research in the area of lightweight cryptography for the Internet of Things is very active — with frequent development of new lightweight ciphers, it is still at a very primitive level. Therefore, it is important that researchers and scientists alike cooperate in developing cryptographic ciphers that are robust against possible security breaches.

1.4 Contributions of Thesis

My goal in this thesis is to strengthen the general knowledge of cryptanalysis, specifically cryptanalysis of lightweight substitution permutation block ciphers. For that reason, I focus on evaluating the strength of two simple block ciphers by performing linear cryptanalysis on them. The discussions available on this thesis can serve as a basic introduction to cryptography and cryptanalysis in embedded systems. The knowledge obtained from this paper can then be extended to help researchers in understanding and designing secure ciphers and systems in the future.

During the course of my research on lightweight cryptography on embedded systems, I realized that the existing literature assumes that the reader has more than basic knowledge of the topics under discussion. As a result, I found it difficult to find and read papers when I was starting research in this area. As my first contribution, I have discussed the key concepts without assuming any knowledge of the concepts beforehand from the reader. Therefore, even new researchers wanting to learn about the cryptography and cryptanalysis in the Internet of Things can read my thesis to prepare themselves to dive deeper into other more extensive research papers.

Additionally, there is active research in cryptanalysis of various new and coming ciphers, but the research fails to provide implementation of these cryptanalysis techniques for readers to better understand the approach and methodologies. The processes involved and the steps taken in performing cryptanalysis are also not discussed in detail. As my second contribution, I have provided an implementation of linear cryptanalysis of a Substitution Permutation Network cipher and the block cipher PRESENT. This implementation will help the reader understand the linear cryptanalysis technique better as the reader can easily follow the implementation together with the discussion. The implementation, written in Python, is the first step towards an automated and customizable linear cryptanalysis that can be improved for a stronger cryptanalysis or extended for other cryptanalysis techniques.

1.5 Outline of Thesis

- In Chapter 2, I introduce the reader to a number of topics and concepts that are imperative in understanding the rest of the thesis. This chapter also discusses the existing works on lightweight cryptography and cryptanalysis present in the literature.

- Chapter 3 discusses the limitations of the current state-of-the-art in providing a sound implementation of linear cryptanalysis for block ciphers. Then, it provides specifics of my implementation approach and methodologies, which complements the Python implementation released with this paper.

- Chapter 4 summarizes the contributions made in this thesis in terms of helping strengthen the general understanding of linear cryptanalysis of Substitution-Permutation Networks. It then outlines future areas of research that are possible.

Chapter 2

BACKGROUND

My thesis is focused on linear cryptanalysis of lightweight substitution-permutation ciphers. This topic generally entails a number of key terms and concepts that are not straightforward and it helps to provide an overall background of different parts of the topic. It is also beneficial to the reader to clarify what kind of domains I am referring to for these ciphers to be used in. I will provide a brief introduction to embedded systems and go on to discuss cryptography and cryptanalysis and conclude this chapter by discussing the cipher I will be using for cryptanalysis.

2.1 Embedded system

An embedded system is a combination of hardware and software designed for a specific purpose. It is generally embedded into larger computing systems and forms a part of a complete system [4]. Typical examples of embedded systems include cellular phones, mobile cameras, GPS sensors etc. Modern automobiles contain many small embedded sensors that continuously monitor and control the operations of a vehicle.

Embedded computing systems are generally small in size, which comes at a cost. The small size necessitates that the power consumption of these devices be kept at a minimum because they are generally incapable of withstanding high heat and it is also difficult to dissipate heat due to the small surface area. Therefore, the devices cannot be fed with data and computations that require tremendous amount of energy consumption. Another important property to note about an embedded system is that their tasks are mostly time-constrained and require high accuracy [5]. For example, sensors used in a car or a plane are responsible for performing their tasks in real-time with great accuracy or it could result in a serious consequence.

Therefore, it is integral that embedded systems perform their specified functions without failure. However, it is not always the case that it is the system's failure to perform a task. Often, there is some form of an external intrusion that causes the system to fail in its task. Similarly, with the Internet of Things becoming increasingly intricate, the embedded systems often have to communicate with one another, which involves sharing a large amount of data that contain valuable and private information about individuals and companies. As is so common, there will, without doubt, be eavesdroppers trying to listen in on these communications between systems. It is absolutely not right for this to be allowed to happen. Therefore, it is imperative that the communication between these systems be secured to the maximum. Cryptography is a solution for this problem, which I will explain in the next section.

2.2 Cryptography

Cryptography is a study of mathematical techniques and algorithms that keep messages secure by making data accessible to and modifiable by only authorized users [6]. Cryptography also ensures that the origin of the data can be authenticated and not repudiated by users. Cryptographic algorithms are used to encrypt and decrypt information with the use of keys to achieve the services mentioned above. There are a number of conventional cryptographic algorithms that would serve the purposes mentioned above without difficulty. However, most of the conventional algorithms perform complex operations on numbers, and this is not ideal for an embedded system because complex operations involve high computations, which results in increased power consumption. Therefore, a recent development in cryptography is the use of algorithms or protocols that are tailored specifically for use in resource-constrained environments like sensors, smart cards and actuators [7]. For the scope of my work, I focus my attention on such cryptographic algorithms, i.e, lightweight cryptographic algorithms.

2.3 Lightweight cryptography

Lightweight cryptography has been frequently cited as an approach that can provide a sufficient level of security without requiring the devices to compromise on speed and power [7]. As the name suggests, lightweight cryptography provides algorithms or protocols that are lightweight, i.e, they work efficiently without requiring the computations necessary of conventional cryptographic algorithms. Hence, lightweight cryptography helps secure the resource constrained embedded computing devices better in comparison to conventional cryptographic algorithms.

As like conventional cryptography, lightweight cryptographic algorithms are mainly divided into two classes: Public Key Algorithms and Private Key Algorithms. Figure 2.1 demonstrates this classification hierarchy of lightweight cryptography.

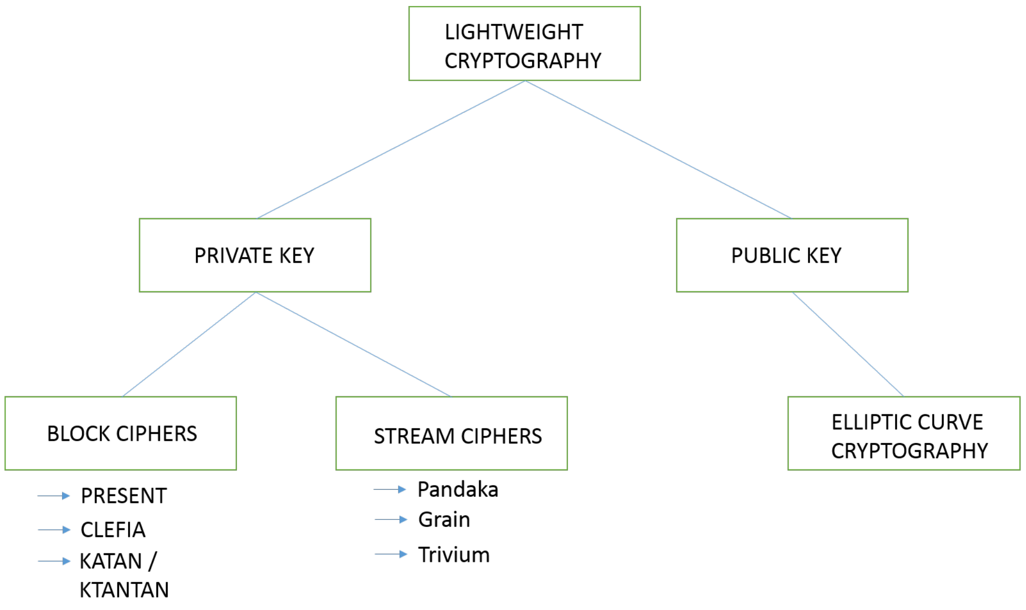

Figure 2.1: Classification hierarchy of Lightweight Cryptography

2.3.1 Public Key Cryptography

Public key or asymmetric key cryptography involves the use of pairs of private and public keys. There are a set of public keys that are easily available to the public and the sender and receiver have their own private keys that they use with the public key to encrypt and decrypt messages. This is made possible by Diffie and Hellman's realization that there are functions that are easy to calculate but very difficult to invert without extra information [8]. The use of a public key actually serves as an advantage because the communicating parties need not share their private key with one another. Public key algorithms operate by performing complex arithmetic operations on large numbers and benefit from the fact that it is very difficult for an intruder to derive the keys in reasonable time. Despite their potential strength, most public key algorithms are computationally very expensive and hence are not suitable for most embedded applications. Elliptic Curve Cryptography is the only public key cryptographic method that has shown potential in embedded systems because of reduced processing needs [6].

2.3.2 Private Key Algorithms

Also called symmetric key algorithms, private key algorithms differ from Public Key algorithms because they use the same key for encryption and decryption. Hence, the keys have to be kept confidential between the sender and the receiver of data. Symmetric key algorithms are often considered to be very quick and are used commonly for encryption and decryption [9].

Symmetric key algorithms are generally sub-divided into block and stream ciphers, which differ distinctly in the key generation. **Block ciphers** partition the plain text into chunks or blocks of data and work with a relatively large partition of data. They take a k-bit block of plaintext as input, and output a corresponding a k-bit block of ciphertext during the encryption process [10]. The decryption process is handled similarly. On the other hand, **stream ciphers** generate a pseudo-random key stream and use it to encrypt the plaintext one bit at a time typically using the bit-wise XOR operation. For this reason, there are a number of differences between a stream cipher cryptographic algorithm and a block cipher, even though they might act similarly on certain modes of operation. Stream ciphers generally run faster than block cipher algorithms and are generally preferred over block ciphers for applications with less computational resources like small embedded devices [11]. Since stream ciphers work with one bit at a time, they have relatively low memory requirements compared to block ciphers. Stream ciphers are also noted to have drawbacks because of the lengthy initialization phase before the first usage. However, there is not a general consensus on which of the two ciphers is better.

The remainder of this thesis focuses on block ciphers and block cipher cryptanalysis. We are now going to look at block ciphers in more detail.

2.3.3 Block Ciphers

A block cipher is a function that takes as inputs a k-bit key and an n-bit string and outputs an n-bit string. Usually, block ciphers consist of a sequence of operations that are run on the n-bit string (or *plaintext*) for a number of rounds. Each round generates a key that is called the *roundkey*. The first round takes the n-bit *plaintext* and the last round returns the n-bit *ciphertext* after r rounds [8]. The round functions need to be invertible because it is important for the receiver of the *ciphertext* to be able to decrypt it into *plaintext*. Therefore the functions that are used for encryption are generally Substitution-Permutation Networks. Substitution adds confusion into the cipher with the use of *substitutionboxes*. The substitution layer possesses non-linear properties, and the large lookup tables required for implementing them require a large amount of memory resources and are difficult to invert [12]. Permutation adds diffusion to the cipher by dissipating the redundancy of the plaintext and by spreading the plaintext over the ciphertext. Even though a number of different linear operations can be combined to make the computations more complicated, the fact that permutations are linear operations makes the permutation operation easily invertible. Hence, ciphers relying solely on diffusion can be broken with minimal effort.

2.3.3.1 Substitution-Permutation Cipher

This section describes the components of a Substitution Permutation Network (SPN), which will be the primary focus of the following chapter. An SPN is a block cipher that consists of R rounds ($R > 1$). If N is the size of the input block to the cipher, then an R round cipher needs R+1 N-bit subkeys. Each round makes use of the round subkeys for the operations whereas the $(R + 1)^{th}$ subkey is required in the end to produce the ciphertext. Each round of the network consists of three layers during encryption. The key mixing layer consists of an XOR operation between the data block input for that round with the round key. The substitution layer takes the input block (of size N) and divides it into M sub-blocks of size n ($N = Mn$). Each of these sub-block forms an input to a $n \times n$ S-Box. This is the non-linear part of the cipher. The permutation layer then consists of a linear transformation of the bits. Since this is a linear mapping, the final round does not usually contain a permutation layer as it can be easily reversed and ,therefore, adds no security. The final $(R + 1)^{th}$ subkey is XOR'd with the output from the round R to form the ciphertext. Decryption in an SPN follows the same operations in the reverse order.

Figure 2.2 shows the first round of an SPN with a 4-bit block size. The input to the S-Boxes are 2-bits.

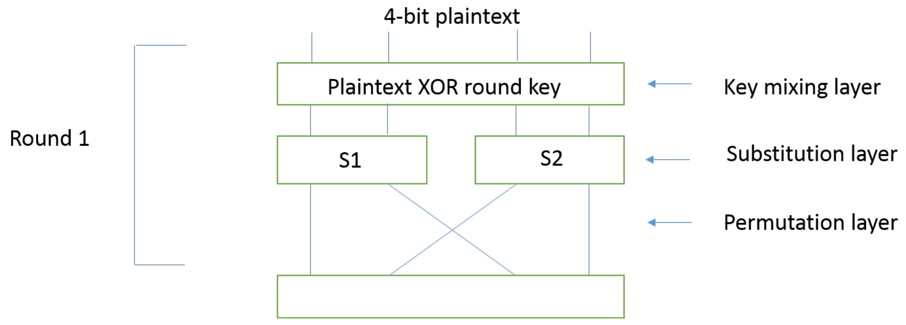

Figure 2.2: Round-1 of an SPN

2.3.3.2 PRESENT

PRESENT is a block cipher designed by Bogdanov et al. [13] with lightweight cryptography and hardware efficiency in mind. It is an example of a Substitution Permutation Network that consists of 31 rounds. PRESENT has an input block length of 64 bits and the key sizes supported are 80 bits and 128 bits. I am going to work with the 80-bit version in this thesis. PRESENT was designed with hardware efficiency in mind, and has also been adopted by the ISO/IEC 29192 as suitable for lightweight cryptography [14]. Each round of PRESENT consists of three main layers: key mixing layer, a permutation layer and a non-linear substitution layer. The key mixing layer is a 64-bit XOR operation to generate round keys K_i for $1 \leq i \leq 32$. The substitution layer is a 64-bit non-linear transform that makes use of the 4-bit S-box 16 times in parallel. The permutation layer is a 64-bit linear bit-by-bit permutation.

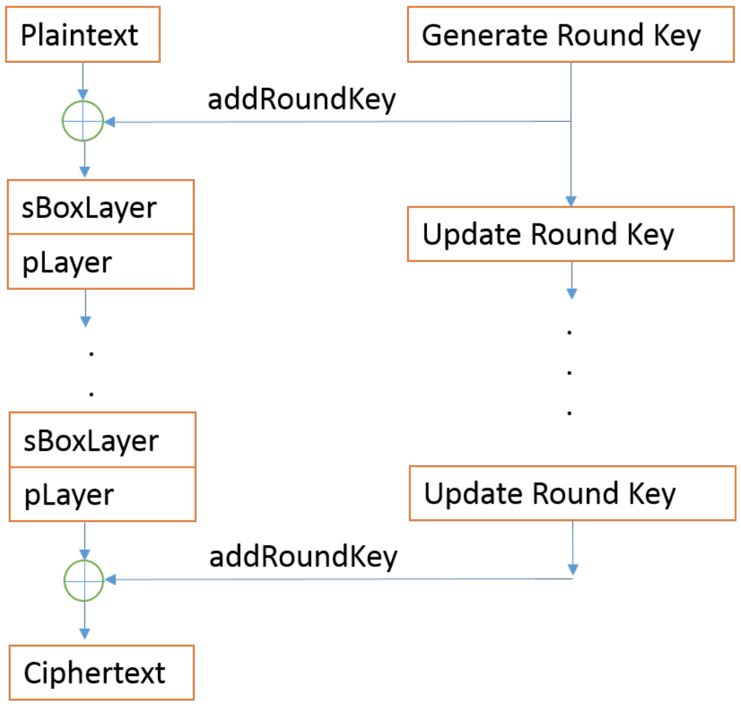

Figure 2.3: Layers of a PRESENT encryption

2.4 Cryptanalysis

While development of new and innovative ciphers is important for preventing new and improved attacks on systems, it is also equally important to ensure that the ciphers that are developed are strong and fail-proof against these attacks. Cryptanalysis is a technique used to make sure that ciphers provide provable security against these attacks. Cryptanalysis is the science of evaluating the strength of cryptographic primitives and protocols [15]. Cryptanalysis is a very common and important approach to study and improve the strength of cryptography. From as early as the development of the Data Encryption Standard, which was one of the revolutions in cryptography, cryptographers have tried to find weaknesses in algorithms [8]. This, in turn, allows strengthening the cryptographic algorithms for better security through cryptography. Cryptanalysis on block ciphers is generally performed by attempting to recover parts of the plaintext or the secret key.

2.4.1 Linear Cryptanalysis

Linear Cryptanalysis is a known plaintext attack, meaning that an intruder gets hold of parts of the plaintext corresponding to the captured ciphertext blocks. Introduced by Mitsuru Matsui in the 1990s, it is one of the most popular cryptanalysis techniques. Linear cryptanalysis uses linear approximations to model non-linear steps in the encryption process. Recovering the key bits or part of the key bits is the final goal of linear cryptanalysis. Linear cryptanalysis studies the evolution of parities of data bits during the encryption process [15]. The goal of linear cryptanalysis is to approximate non-linear operations of a cipher using linear approximations of the form

$$P[i_1, i_2, i_3, ...i_a] \oplus C[j_1, j_2, j_3, ...j_b] = K[k_1, k_2, k_3, ...k_c]$$

where $A[i_1, i_2, i_3, ...i_n]$ represents $A[i_1] \oplus A[i_2] \oplus A[i_3] \oplus ... \oplus A[i_n]$ for a set A and $A[i]$ represents the $i - th$ bit of A [16]. The sets P, C and K represent the plaintext, ciphertext and the key bits respectively. The equation represents the XOR "sum" of a plaintext bits and b ciphertext bits.

If we assume that an approximation holds with a probability p, the cipher is very difficult to break with linear cryptanalysis if $p = \frac{1}{2}$ because that gives an indication of a perfectly random cipher without any weaknesses of linearity. Therefore, linear cryptanalysis works the best when the probability bias, denoted by $|p - \frac{1}{2}|$, is large enough, i.e, the probability is far from being random).

2.4.2 Finding the best linear expression

One of the most important obstacles to tackle in linear cryptanalysis is finding the best linear expression. Since the S-boxes are the only non-linear components of an SPN, the properties of the S-boxes are to be considered to figure out the best linear expressions for approximation. By looking at all possible inputs and outputs to an S-Box, linear approximations can be developed, and these approximations can be used to find a linear expression with a large enough bias.

2.4.3 Piling-up Lemma

To find the probability bias of a linear expression, cryptanalysts use what is called the piling-up lemma. Since looking at all possible combinations of plaintexts and ciphertexts is impossible during cryptanalysis, this lemma is used to approximate the probability of a linear expression [16]. The lemma is stated as follows:

Lemma: Given n independent random variables $X_1, X_2, X_3, ...X_n$ taking on values from {0,1}, then the bias $\epsilon = p - \frac{1}{2}$ of the sum $X = X_1 \oplus X_2 \oplus X_3 \oplus ... \oplus X_n$ is given by:

$$\epsilon = 2^{n-1} \prod_{j=1}^{n} \epsilon_j \tag{2.1}$$

where ϵ_1, ϵ_2, ϵ_3, ... ϵ_n are the biases of the terms $X_1, X_2, X_3, ...X_n$.

Therefore, this lemma is used to combine the individuals biases of the terms to calculate an approximate of the probability of linear expression, which gives an indication of how well linear cryptanalysis would work for a given expression or if it would work.

In the next section, I will use this lemma to find a linear expression that can be used to perform linear cryptanalysis on a Sustitution-Permutation Network of 4 rounds.

Chapter 3

LICS : LINEAR CRYPTANALYSIS OF SUBSTITUTION PERMUTATION NETWORK

After discussing the key concepts and ideas necessary for understanding linear cryptanalysis, I am now going to focus attention on a specific Substitution-Permutation Network (SPN) cipher and guide the reader through the process of implementing linear cryptanalysis on the cipher. This chapter summarises the motivations on why an efficient linear cryptanalysis technique is helpful for cryptanalysis, and points the limitations of the current state-of-the-art in helping implement linear cryptanalysis. Then, it provides an open source automated and customizable framework for implementing linear cryptanalysis on a Substitution-Permutation Network cipher.

3.1 Motivation

The research for new and efficient methods of lightweight cryptography for use in embedded systems is growing rapidly. However, despite increasing popularity of these cryptographic techniques amongst researchers, the understanding of these techniques is still beyond clear. The existing literature summarises definitions and results of these techniques without really going into detail about the processes involved and the steps taken in performing cryptanalysis. However, understanding various cryptanalysis techniques is imperative to cryptography to make sure that new ciphers that are developed provide provable security against these attacks. Therefore, my goal in this paper is to make the process of performing linear cryptanalysis of an SPN as clear as possible so that it can help in understanding and implementing improved linear and other forms of cryptanalysis.

3.2 Related Work

There have been many block ciphers that have been developed recently with hardware and software efficiency in mind. Hong et al. [17] proposed a low-resource block cipher called HIGHT with hardware efficiency in mind. They claimed that it was proper for use in sensors and RFID tags. The authors claim that differential and linear cryptanalysis of HIGHT are not sufficiently efficient, claiming that it provides enough security for use in embedded systems. Apart from that, there does not seem to be a whole lot of research done on the efficiency of HIGHT against cryptanalysis techniques.

Because of its increasing popularity in the realms of lightweight cryptography, cryptanalysis of PRESENT has been massively performed. Even though the designers of PRESENT have claimed that differential and linear cryptanalysis of PRESENT are practically covered in their design [13], cryptanalysts have continued to perform variations of differential and linear cryptanalysis on PRESENT. Wang [18] claims that a 16-round PRESENT can be broken using 2^{64} chosen plaintexts and 2^{64} memory accesses. Collard and Standaert [14] propose a statistical saturation attack that exploits a cipher by exploring a generic way to find partitions for a given cipher that can lead to efficient attacks. The name statistical saturation attack refers to the fact that it exploits the weaknesses in the diffusion properties of a cipher. PRESENT has a particular weakness in its diffusion layer and hence is a good target for the proposed statistical saturation attack. Ohkuma [19] has shown, through the use of linear single-bit paths, that the linear deviation of paths can be higher than what the designers of PRESENT initially claimed. They call the portions of keys where this happens weak keys and hence they show that linear cryptanalysis is possible with $2^{63.5}$ known plaintexts for 32% of the keys.

3.3 Limitations of Current Work

Despite significant amount of research on different block ciphers and substitution-permutation networks, there does not exist an easy-to-understand implementation of linear cryptanalysis. Except for the tutorial presented by Heys [20], there does not seem to be a work of literature that helps in understanding the process. Even their tutorial is made difficult to follow because of the lack of implementation code and the fact that their discussion is only valid for a particular SPN structure that they define in the paper.

3.4 Goals

As a result of the lack of a research paper that explains the process of linear cryptanalysis to even beginners doing research on cryptanalysis, I have produced this comprehensive paper that provides the background of important concepts (in Chapter 2) and explains the whole process of linear cryptanalysis on an SPN. More significantly, I will release an open source framework for implementing linear cryptanalysis that helps a great deal in understanding a general 4-round linear cryptanalysis. This framework will be customizable so that it can be improved upon to be used for improved cryptanalysis technique as well as other forms of cryptanalysis. I use the Python programming language to implement linear cryptanalysis of an SPN for up to 4 rounds. Python is a particularly good choice for the purpose because it is one of the most popular languages used by researchers nowadays, making sure that it will be useful for others to build upon. The implementation is also very easy to understand, even for people with not much knowledge about programming, because it is a very high-level language.

3.5 SPN Cipher

Before discussing the cryptanalysis, I am going to separate the SPN cipher that I am using into its components. The SPN chosen is a 4 round cipher that takes a 16-bit block of input. The encryption process of this SPN has three layers: substitution, permutation and the key mixing layer. The substitution takes place using a 4×4 S-box that is used 4 times in parallel. The 16-bit data block is broken into four 4-bit sub-blocks which form the input to the S-Box. Substitution, as mentioned above, is a non-linear mapping. The S-Box used for this paper is shown in Table 3.1. It should be noted that this is just an example of an S-Box and could vary depending on the size and structure of the SPN.

Table 3.1: Representation of an S-Box

Input(x)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output S(x)	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Since we are using a 4×4 S-Box 4 times in parallel, if an input to the cipher was a 16-bit value like $[1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]$, after the substitution part, the resulting output would be $[0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0]$. The entire input is divided into 4 4-bit values and then the integer equivalent of the 4-bit values are substituted with their corresponding outputs from the table.

The permutation part of the round involves transposition of the bits. It is a symmetric permutation defined as in Table 3.2. It is a simple permutation where the output i of S-Box j is connected to the input j of S-Box i. Figure 3.1 demonstrates the effects of the permutation layer.

Table 3.2: Representation of permutation

Input(x)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

The third layer of the SP network simply involves an XOR operation between the data block for that round and the round key. If the input data block to this layer is $[1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0]$ and the round key for that round is $[0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1$
then the output from the key mixing layer is $[1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0]$. This is just the result of XOR-ing individual bits of the data block and the round key.

Figure 3.1 demonstrates the 1 round encryption of a 16-bit plaintext in the mentioned cipher.

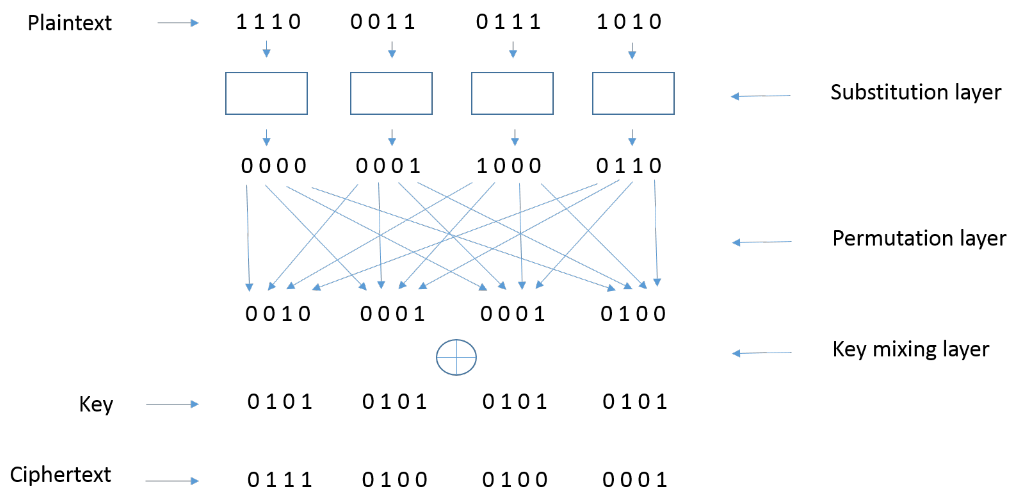

Figure 3.1: Encryption example for a single round

Now that we understand the different components of a SPN, we are going to look at how to analyse these components to perform linear cryptanalysis on the SPN.

3.6 Overview

The process of linear cryptanalysis of a cipher requires a number of steps from understanding the cipher to carefully approximating paths to exploit the linearity (or the deviation from perfect randomness) of the cipher. Figure 3.2 demonstrates the process.

3.6.1 Generating a Linear Approximation Table

The linear approximation table is the first step that provides an indication of where the cipher is weak and can be exploited. To construct a linear approximation table, all inputs and outputs to the S-Boxes should be enumerated. Since our S-boxes are 4x4, all possible inputs and outputs can be enumerated using a 16x16 linear approximation table. The linear approximation table gives values for the probabilities of equations involving all possible inputs and outputs holding true. That is, all linear

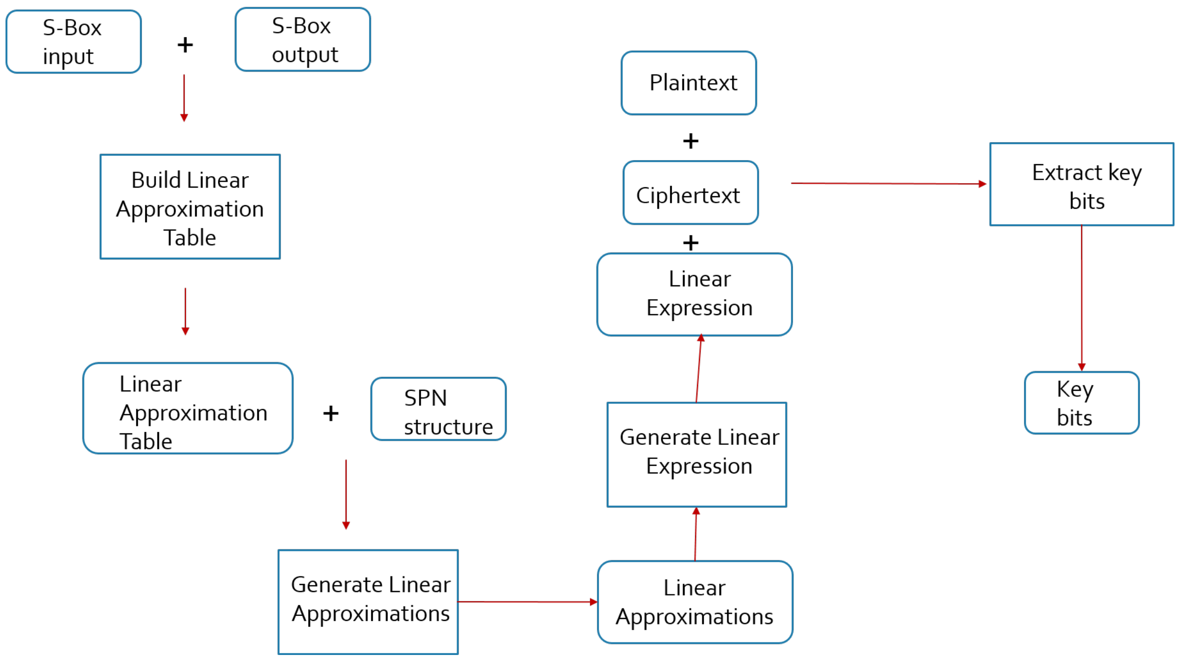

Figure 3.2: Framework for linear cryptanalysis

equations that can be formed using the 4 bits as inputs and 4 bits as outputs are encompassed in this table. Whenever the probabilities of these equations are high, it gives an indication that these equations could potentially be used to form approximations to exploit for linear cryptanalysis.

3.6.2 Generating Linear Approximations

The information from the linear approximation table can help find the necessary linear approximations by looking at the probabilities of the linear equations holding true. The only other information needed in coming up with these approximations are the active S-Boxes for each round. When combined together, the active S-Boxes and the linear equations form the linear approximations. The approximations are formed such that they result into an expression that takes place with a high probability. The current framework generates linear approximations using a random approach. It goes through the S-Boxes of the first round to choose random inputs and outputs for the first round. Thereafter, the inputs for the consecutive rounds are the results of the outputs of the previous rounds passed through the Permutation boxes. It is necessary to check, during random approximation generation, to make sure that whenever there is an input to an S-Box, there is an output too and vice-versa. This check also is also sufficient to ensure that there is always a path that reaches the final round.

An example approximation generated is as below:

$$S_{12} : X_2 = Y_0 \oplus Y_1$$

$$S_{13} : X_0 \oplus X_2 = Y_0$$

$$S_{14} : X_0 \oplus X_1 \oplus X_2 = Y_0 \oplus Y_1$$

$$S_{21} : X_1 \oplus X_2 \oplus X_3 = Y_0 \oplus Y_1$$

$$S_{22} : X_1 \oplus X_3 = Y_0$$

$$S_{31} : X_0 \oplus X_1 = Y_0 \oplus Y_3$$

$$S_{32} : X_0 = Y_3$$

3.6.3 Generating a Linear Expression

Now that we know how to generate the approximations, the approximations are combined together to generate a linear expression. Combining the approximations entails following the path taken by the equations and the S-Boxes to generate the linear expression and its respective probability. Provided with the linear approximations, the current framework automatically generates a linear expression. That is, when linear approximations involving the S-boxes and the inputs to and outputs from them are given, a linear expression is automatically generated that can be used to perform linear cryptanalysis. In addition, the framework also checks to ensure that the linear expression generated is valid by going through the individual approximations and following the paths specified by the approximations.

3.6.4 Extracting Key Bits

After generating the linear expression, it is finally used to extract key bits. As a matter of fact, the linear expression generated consists of output from the second-last round of the cipher. With the help of possible key bits and pairs of plaintext and ciphertext, an expression is used to check how well a certain key bit does in the key mixing layer to form an equation that holds with a probability that is the farthest from $\frac{1}{2}$ (farthest from randomness). That key bit is then decided to be the actual key bit. This process of extracting key bits is described in more detail in the next section.

The following section illustrates the process of linear cryptanalysis of an SPN using specific approximations.

3.7 Linear Cryptanalysis of an SPN

A detailed discussion of the linear cryptanalysis technique for the particular SPN is available in Heys [20]. I will describe the process of extracting the key bits at a very high level in this paper. The implementation of linear cryptanalysis provided with this paper should be read alongside this paper to have a better understanding of the whole process of linear cryptanalysis.

For the sake of consistency and ease, I use the same approximations provided in Heys [20]. Due to the lack of time, the current implementation does not generate usable linear approximations in the sense that those approximations do not lead to good linear expressions. In the next chapter, I have outlined the work that could be done to generate good linear approximations.

The approximations that I use are:

$$S_{12} : X_1 \oplus X_3 \oplus X_4 = Y_2$$

$$S_{22} : X_2 = Y_2 \oplus Y_4$$

$$S_{32} : X_2 = Y_2 \oplus Y_4$$

$$S_{34} : X_2 = Y_2 \oplus Y_4$$

These approximations can then be combined or concatenated together to get a linear expression by following the structure of the SPN and the active S-Boxes. The structure and the path followed by the approximation can be seen in Figure 3.3. Then, we can generate an expression that consists of only plaintext bits and the data bits from the second last round of S-boxes. The generated expression is as follows:

$$U_{4,5} \oplus U_{4,7} \oplus U_{4,13} \oplus U_{4,15} \oplus P_4 \oplus P_6 \oplus P_7 = 0$$

where each P_i are the i-th bit value of a plaintext P and $U_{j,k}$ is the k-th bit input value to the corresponding S-Box in the j-th round.

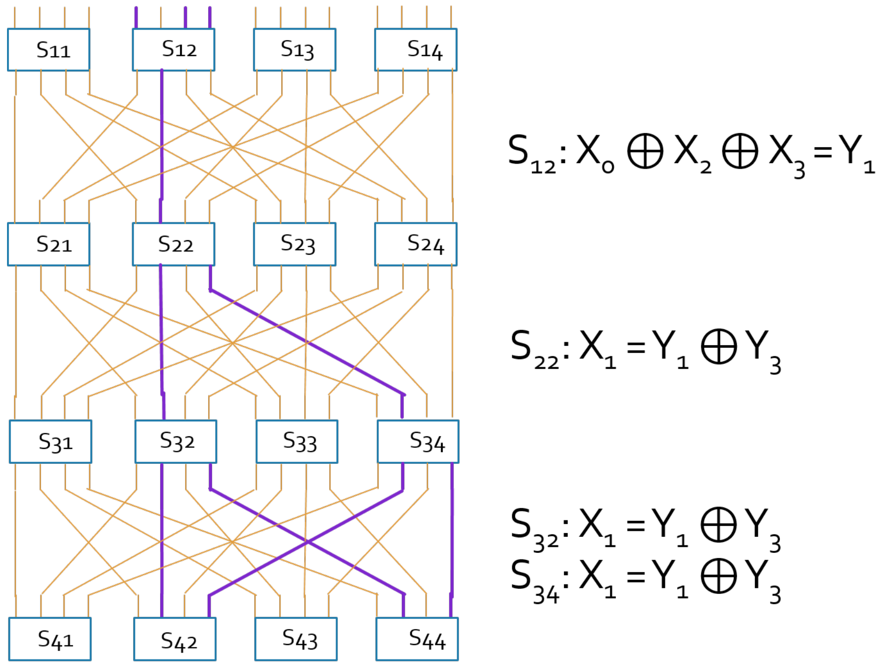
S11
S12
S13
S14
$S_{12}: X_0 \oplus X_2 \oplus X_3 = Y_1$
S21
S22
S23
S24
$S_{22}: X_1 = Y_1 \oplus Y_3$
S31
S32
S33
S34
$S_{32}: X_1 = Y_1 \oplus Y_3$
$S_{34}: X_1 = Y_1 \oplus Y_3$
S41
S42
S43
S44

Figure 3.3: Generating a linear expression

We only get an expression up to the second to last round of the cipher because the approximations provided include S-boxes up to the second to last round. Since we took the output from the second last round (or the input to the last round) of the cipher, the expression that we generated is also an $R-1$ round expression for an R round cipher. Nevertheless, this expression can be used to extract parts of the subkey from the last round. To that extent, first we need to decrypt the last round of the cipher. Since the last round of the cipher does not contain a permutation layer, all we need to do to decrypt the last round is to reverse the process of encryption. The first step in decryption is to XOR the ciphertext bits with the corresponding key bits of the partial round key. This intermediate value is then passed backwards through the S-Box (or pass it through the reverse S-Box).

For the actual linear cryptanalysis, this same process of decryption is performed for a large number of plaintext/ciphertext pairs and each value of the possible subkey bit is given a count. Then, whenever the linear expression that we came up with holds true for the input to the last round of the cipher (which is the result from decryption of the last round) and corresponding bits of the known plaintext, the count for the corresponding subkey bit is incremented. Now, the subkey bit which has a count that differs the maximum from half the number of plaintext/ciphertext pairs is going to be the correct value for the subkey. This is true because that indicates that the linear expression held true with a probability that was of the greatest difference from $\frac{1}{2}$.

3.8 Implementation

I have implemented a linear cryptanalysis attack on the Substitution Permutation Network cipher using 10000 plaintext/ciphertext pairs. A result of one of the executions is shown in Table 3.3. The table shows that the subkey bit pair of 15-1 holds true with a probability bias that is the highest (meaning it differs the maximum from $\frac{1}{2}$.) Additionally, the actual subkey pair for this exection was: [1111, 0001] which is [15,1]. Therefore, the results demonstrate the claim. Following this process, every execution of the attack gives me the correct result for the partial subkey bits, which means that the Substitution-Permutation Network can be broken very efficiently upto 4 rounds. The link to the implementation can be found in my github repository at: https://github.com/acharyab15/spn_linear_cryptanalysis.git

Table 3.3: Subkey pairs and their corresponding probability bias

Pair	Probability Bias
15 - 1	0.034300
14 - 1	0.032600
15 - 0	0.027500
15 - 15	0.025000
2 - 12	0.022900
3 - 12	0.019400
15 - 14	0.019200

3.9 Discussion

This chapter discusses a successful implementation of linear cryptanalysis on a 4-round SPN. The success rate of this implementation is 100% because of the fact that the cipher is a 4-round SPN with 16-bit inputs. The rate of success decreases as the number of rounds and the size of the input decreases because the approximations start becoming weaker. However, understanding my implementation of linear cryptanalysis helps in developing the background required for an improved understanding of cryptanalysis techniques.

Unlike most works of research, I have provided an open source Python framework that automates various steps of linear cryptanalysis and makes it easily accessible to readers. The framework automatically generates linear expressions from approximations that can be used for linear cryptanalysis. Linear approximations can also be randomly generated, however, the current implementation needs some work before it can generate good linear approximations for cryptanalysis. The structure of the framework is very customizable so that with some slight changes to the framework, improved cryptanalysis can be performed on increased rounds of the cipher. With some work, it can also be extended to other kinds of cryptanalysis like differential cryptanalysis on the cipher.

Chapter 4
CONCLUSION

This thesis deals with linear cryptanalysis of Substitution-Permutation Networks. It is a first step towards developing an open source framework for an automated and customizable linear cryptanalysis of SPNs.

4.1 Contributions

The contributions of my work presented in this thesis can be summarized in two major points:

1. A thorough examination of the existing literature to strengthen the general understanding of lightweight cryptography, especially block ciphers. Through careful assessment of two substitution permutation block ciphers, I have elucidated the reader to the different components of a block cipher and the various processes that are undertaken in encrypting and decrypting messages using a substitution-permutation block cipher.

2. An open source framework for implementing linear cryptanalysis of substitution-permutation networks in python has also been provided as a contribution. This framework, which takes the reader from generating linear approximations to extracting the key bits from the cipher, is important in studying the process of linear cryptanalysis. The implementation also generates linear approximations using a random path generation method. Parts of the framework have

also been automated and work has been done to make this framework customizable so that it can be used with other ciphers and be improved for a more extensive cryptanalysis. The implementation of the framework can be found in: https://github.com/acharyab15/spn_linear_cryptanalysis.git

4.2 Future Work

The work presented in this research can give directions to different kinds of future research:

1. The task of generating linear approximations is a difficult one which needs a great deal of understanding. The current implementation generates linear approximations using a random path generator. However, that is not likely to result in approximations that lead to expressions with high probability bias. So work still needs to be done in finding a way to approximate paths that results into the best linear expression.

2. The existing literature talks about a ”big enough probability bias” for linear cryptanalysis to work efficiently. However, it is not clear how big is big enough. Research can be done to calculate how much bias is needed for certain rounds for linear cryptanalysis to work efficiently.

3. The current work only looks at Substitution-Permutation network ciphers. The discussion and implementation provided can be built upon to look at other block ciphers, and also be extended to stream ciphers with some work.

4. Linear cryptanalysis is the only cryptanalysis technique studied in this paper. During the course of my research, I came across the processes of performing of differential cryptanalysis too. The knowledge of linear cryptanalysis from this paper can definitely be extended to study differential cryptanalysis in greater detail.

Bibliography

[1] Rob Van Kranenburg, Erin Anzelmo, Alessandro Bassi, Dan Caprio, Sean Dodson, and Matt Ratto. The internet of things. *A critique of ambient technology and the all-seeing network of RFID, Network Notebooks*, 2, 2011.

[2] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1, 2011.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[4] S. Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*, 6(4):440–459, 2014.

[5] Michael Barr and Anthony Basa. *Programming Embedded Systems: with C and GNU Development Tools*. O'Reilly Media Inc., Connecticut, USA, 2006.

[6] Thomas Wollinger, Jorge Guajardo, and Christof Paar. Cryptography in embedded systems: An overview. *Proc. Embedded World 2003*, pages 735–744, 2003.

[7] Masanobu Katagi and Shiho Moriai. Lightweight cryptography for the internet of things. *Sony Corporation*, pages 7–10, 2008.

[8] Christophe De Canniere, Alex Biryukov, and Bart Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356, 2006.

[9] Ayushi. A symmetric key cryptographic algorithm. *International Journal of Computer Applications*, 1(15), 2010.

[10] Bart Preneel and Hongjun Wu. Cryptanalysis and design of stream ciphers. 2008.

[11] Min Chen, Shigang Chen, and Qingjun Xiao. Pandaka: A lightweight cipher for RFID systems. In *INFOCOM, 2014 Proceedings IEEE*, pages 172–180. IEEE, 2014.

[12] Hannes Kruppa and Syed Umair Ahmed Shahy. Differential and linear cryptanalysis in evaluating aes candidate algorithms. 1998.

[13] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007. doi: 10.1007/978-3-540-74735-2_31.

[14] Baudoin Collard and F-X Standaert. A statistical saturation attack against the block cipher present. In *Topics in Cryptology–CT-RSA 2009*, pages 195–210. Springer, 2009.

[15] Elad Pinhas Barkan. Cryptanalysis of ciphers and protocols. *Ph.D. Thesis*, 2006.

[16] Alex Biryukov and Christophe De Cannière. Linear cryptanalysis for block ciphers. In *Encyclopedia of Cryptography and Security*, pages 722–725. Springer, 2011.

[17] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, et al. Hight: a new block cipher suitable for low-resource device. In *Cryptographic Hardware and Embedded Systems-CHES 2006*, pages 46–59. Springer, 2006.

[18] Meiqin Wang. Differential cryptanalysis of reduced-round present. In *Progress in Cryptology–AFRICACRYPT 2008*, pages 40–49. Springer, 2008.

[19] Kenji Ohkuma. Weak keys of reduced-round present for linear cryptanalysis. In *Selected Areas in Cryptography*, pages 249–265. Springer, 2009.

[20] Howard M Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.