Linear Cryptanalysis of SPN Cipher

By
Tabassum Fatima and YashVardhan
IIIT Delhi
CSE546 Applied Cryptography
Instructor - Ravi Anand
Date: *17 November 2024*

Abstract

Linear cryptanalysis is a prominent method for attacking symmetric-key block ciphers, particularly those based on the Substitution-Permutation Network (SPN) architecture. Introduced by Mitsuru Matsui in 1993, this technique exploits linear approximations between plaintext bits, ciphertext bits, and key bits to recover secret keys. This report provides a comprehensive overview of linear cryptanalysis, detailing its historical significance, the structure and functioning of SPN ciphers, and the methodology for key recovery through linear attacks. By implementing a simple SPN cipher and demonstrating the attack process, this report aims to enhance understanding of both the cryptanalytic techniques and their implications for cipher design.

1. **Introduction**

The Substition-Permutation Network (SPN) cipher is a symmetric-key block cipher design that transforms plaintext into ciphertext through a series of well-defined operations. This structure is characterized by its use of substitution and permutation layers, which contribute to the security of the encryption process. SPN ciphers are widely recognized for their robust security properties and are foundational in modern cryptography systems.

SPN (Substitution-Permutation Network) ciphers are widely utilized in various domains due to their robust security features and efficiency. They form the foundation for several cryptographic standards, including the Advanced Encryption Standard (AES). Rijndael, the algorithm selected as AES, is based on a modified SPN structure and is globally used for secure data transmission. Many block ciphers also employ SPN designs because of their balance between security and performance.

SPN ciphers also play a crucial role in ensuring confidentiality and integrity in secure communication. They are integrated into messaging applications to safeguard user data and protect sensitive information, such as credit card numbers, during e-commerce transactions. Moreover, SPN ciphers are well-suited for embedded systems, where resource constraints are a significant concern. They are implemented in Internet of Things (IoT) devices to ensure secure communication and data protection. Similarly, smart cards, commonly used in payment systems and identification cards, leverage SPN ciphers for their efficiency in environments with limited computational resources.

Linear cryptanalysis has evolved into one of the most effective methods for evaluating the security of symmetric key block ciphers. The technique relies on identifying linear relationships within the cipher's operations, allowing attackers to recover secret keys with fewer known plaintext-ciphertext pairs than previously required. This report focuses on applying linear cryptanalysis to a basic SPN cipher, illustrating both the theoretical foundations and practical implementations of this attack strategy.

2. Detailed Description of the Encryption Process in an SPN Cipher

The encryption process in a Substitution-Permutation Network (SPN) cipher is a structured and iterative method that transforms plaintext into ciphertext through multiple rounds of operations. This section provides an in-depth look at each component of the encryption process, including S-box substitution, permutation, key mixing, and the overall encryption function.

2.1 Structure of SPN Cipher

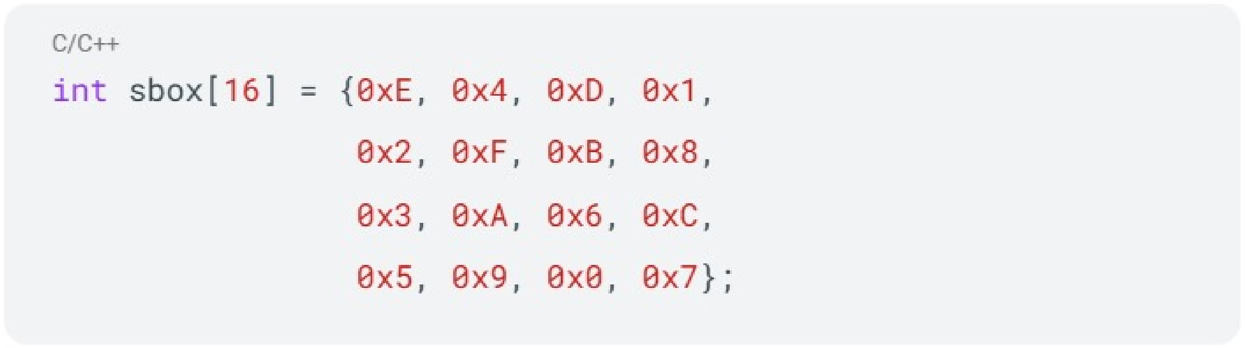
An SPN cipher operates on fixed-size blocks of data, typically 16 bits, and employs a series of rounds (commonly four or more) to achieve strong diffusion and confusion properties. Each round consists of three main operations:

- Substitution: Nonlinear transformation using S-boxes.
- Permutation: Rearrangement of bits to enhance diffusion.
- Key Mixing: Incorporation of subkeys through XOR operations.

2.2 Components of the Encryption Process

2.2.1 S-box Substitution

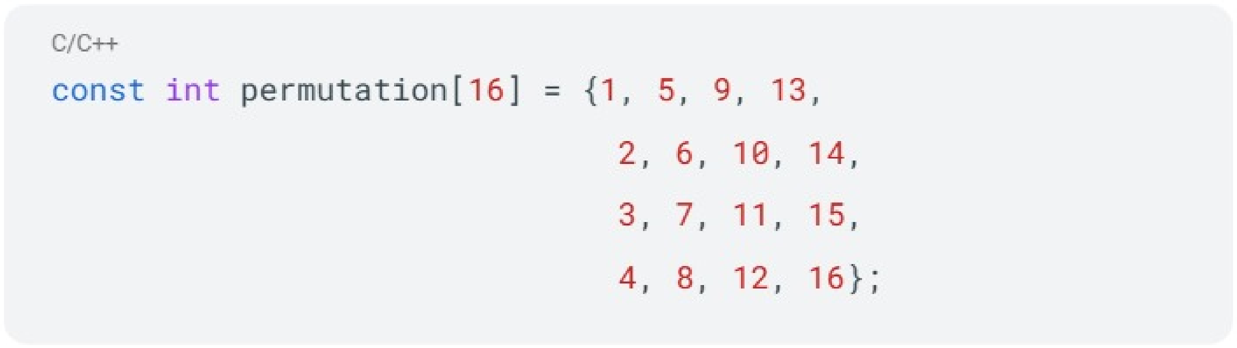
The S-box is a crucial element in the SPN structure that provides nonlinearity to the cipher. Each 4-bit input is mapped to a unique 4-bit output using a predefined table. This S-box is applied to the state during each round by individually processing each nibble (4 bits). In the table, the hexadecimal notation's most significant bit represents the S-box's leftmost bit in the following code.



```
C/C++
int sbox[16] = {0xE, 0x4, 0xD, 0x1,
                0x2, 0xF, 0xB, 0x8,
                0x3, 0xA, 0x6, 0xC,
                0x5, 0x9, 0x0, 0x7};
```

2.2.2 Permutation

The permutation layer further rearranges the bits after substitution to diffuse individual bits' influence across the output. The permutation is defined by a fixed mapping specified in the code:



```
C/C++
const int permutation[16] = {1, 5, 9, 13,
                             2, 6, 10, 14,
                             3, 7, 11, 15,
                             4, 8, 12, 16};
```

2.2.3 Key Mixing

Key Mixing is performed by XORing the current state with a subkey derived from a master key. This operation enhances security by ensuring that each round's output depends on the plaintext and the key. Subkeys are generated from a master key through a critical schedule process.

In the implementation, five subkeys are derived from an input master key.

2.3 Encryption

2.3.1 Key Structure

Master Key and Round Keys:

- The SPN cipher utilizes an 80-bit master key, which is crucial for generating the subkeys used during encryption.
- From this master key, five 16-bit round keys are derived. Each round key is essential for key mixing during each round of encryption, contributing to the overall security of the cipher.

2.3.2 Round Function

The encryption process consists of a series of operations performed in multiple rounds. In this implementation, there are four main rounds followed by a final key mixing operation. Each round includes three primary operations:

Key Mixing

Key mixing is performed at the beginning of each round by XORing the current state with a round-specific subkey. This operation ensures that the output of each round is influenced by the secret key, enhancing security against cryptanalysis. The mixing function can be represented as:

$$\mathbf{State_{new} = State_{old} \oplus Subkey}$$

Substitution

In each round, after key mixing, a substitution operation is applied using an S-box. The S-box provides nonlinearity to the cipher by mapping input bits to different output bits. For example, a 4x4 S-box can transform each 4-bit input into another 4-bit output based on a predefined table. This operation is critical for thwarting linear and differential attacks. The substitution operation can be expressed as:

$$\mathbf{Substituted\ State{=}S(State)}$$

where S denotes the S-box transformation.

Permutation

Following substitution, a permutation operation rearranges the bits of the substituted state according to a predefined mapping. This step enhances diffusion by ensuring that changes in one bit of the plaintext affect multiple bits in the ciphertext. The permutation can be represented as:

$$\mathbf{Permuted\ State{=}P(Substituted\ State)}$$

where P denotes the permutation function.

2.3.3 Rounds of Encryption

The encryption process consists of four rounds, where first three rounds follow these steps:

1. Key Mixing: The current state is mixed with a subkey.
2. Substitution: The result undergoes substitution using the S-box.
3. Permutation: The substituted output is then permuted.

The last round differs slightly as it includes key mixing and substitution but omits permutations to finalize the ciphertext structure effectively.

2.3.4 Final State

In the final round:

- The state undergoes one last key mixing with the penultimate subkey.
- Substitution is applied using the S-box.
- Finally, a last key mixing occurs using the final subkey, ensuring that no further permutations are applied.

3. **Linear Cryptanalysis on basic SPN cipher**

Linear cryptanalysis is a statistical attack technique against block ciphers like SPN (Substitution-Permutation Network). It exploits linear approximations to the behaviour of the cipher to recover information about the secret key. Linear cryptanalysis exploits the likelihood of specific linear relationships between plaintext bits, intermediate bits from the second-to-last round output (treated as "ciphertext" bits), and key bits. It is a **known plaintext attack**, meaning the attacker uses a collection of plaintexts and their corresponding ciphertexts, though they cannot choose which plaintexts are available. The plaintexts are assumed to be a random set. The approach involves approximating part of the cipher's behaviour using a linear equation, where the linearity is defined using bitwise XOR (denoted as $\oplus$). A typical linear expression might look like this:

$$\mathrm{Xi1} \oplus \mathrm{Xi2} \oplus \cdots \oplus \mathrm{Xiu} \oplus \mathrm{Yj1} \oplus \mathrm{Yj2} \oplus \cdots \oplus \mathrm{Yjv} = 0 — (1)$$

Here:

- Xi represents bits of the plaintext (input).
- Yj represents bits of the second-to-last round output.
- The equation sums u plaintext bits and v intermediate bits; the result is always 0 (mod 2).

3.1 History and Importance of Linear Cryptanalysis

3.1.1 Historical Background

Mitsuru Matsui first introduced linear cryptanalysis at EUROCRYPT '93 as a theoretical attack against the Data Encryption Standard (DES). The method demonstrated that DES could be attacked effectively using linear approximations, which led to its practical application in breaking DES in subsequent years. The significance of linear cryptanalysis lies in its broad applicability; it has been adapted for various block ciphers beyond DES, influencing the design principles of modern encryption algorithms.

3.1.2 Importance in Cryptography

The relevance of linear cryptanalysis extends beyond its immediate applications in attacking ciphers. It has played a crucial role in shaping the design of contemporary symmetric key algorithms. Many modern ciphers, including Rijndael (the basis for AES), have been designed with countermeasures to thwart linear attacks. Understanding linear cryptanalysis is essential for cryptographers, as it provides insights into potential vulnerabilities and informs secure cipher design.

3.2 Tools Used in Linear Cryptanalysis

Before going into the actual process of conducting linear cryptanalysis, it is crucial to understand the tools and concepts that form its foundation. These tools allow us to identify linear approximations and biases and exploit them to recover key bits. Below is an overview of these essential tools:

3.2.1 Piling-up lemma

The **Piling-Up Lemma** is a fundamental result in linear cryptanalysis. It provides a way to compute the bias of a linear expression involving multiple independent binary variables. The **Piling-Up Lemma** determines the bias of the XOR (exclusive OR) of multiple independent binary random variables. Bias (ϵ) measures how much the probability of a binary variable being 0 or 1 deviates from 0.5. For a single variable X:

$$\text{Bias: } \epsilon = \mathrm{P(X{=}0)} - 0.5$$

When combining n independent binary random variables X1, X2, ..., Xn using XOR ($\oplus$), the lemma states that the overall bias of their XOR, $\mathrm{Z} = \mathrm{X1} \oplus \mathrm{X2} \oplus \cdots \oplus \mathrm{Xn}$, can be computed as the product of their individual biases and $2^{\text{n-1}}$.

$$\epsilon(\mathrm{z}) = \epsilon 1 * \epsilon 2 * \ldots * \epsilon \mathrm{n} * 2^{\text{n-1}}$$

where ϵi is the bias of each variable Xi.

3.2.2 Linear Approximation Tables

Consider the S-box representation of Figure 2 with input X = [X1, X2, X3, X4] and a corresponding output Y = [Y1, Y2, Y3, Y4]. All linear approximations can be examined to determine their usefulness by computing the probability bias for each. Hence, we examine all expressions of equation (1) where X and Y are the S-box input and outputs, respectively. For example, for the S-box used in our cipher, consider the linear expression,

$$X2 \oplus X3 \oplus Y1 \oplus Y3 \oplus Y4 = 0$$

or equivalently,

$$X2 \oplus X3 = Y1 \oplus Y3 \oplus Y4$$

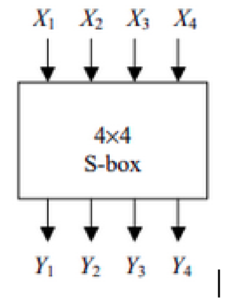
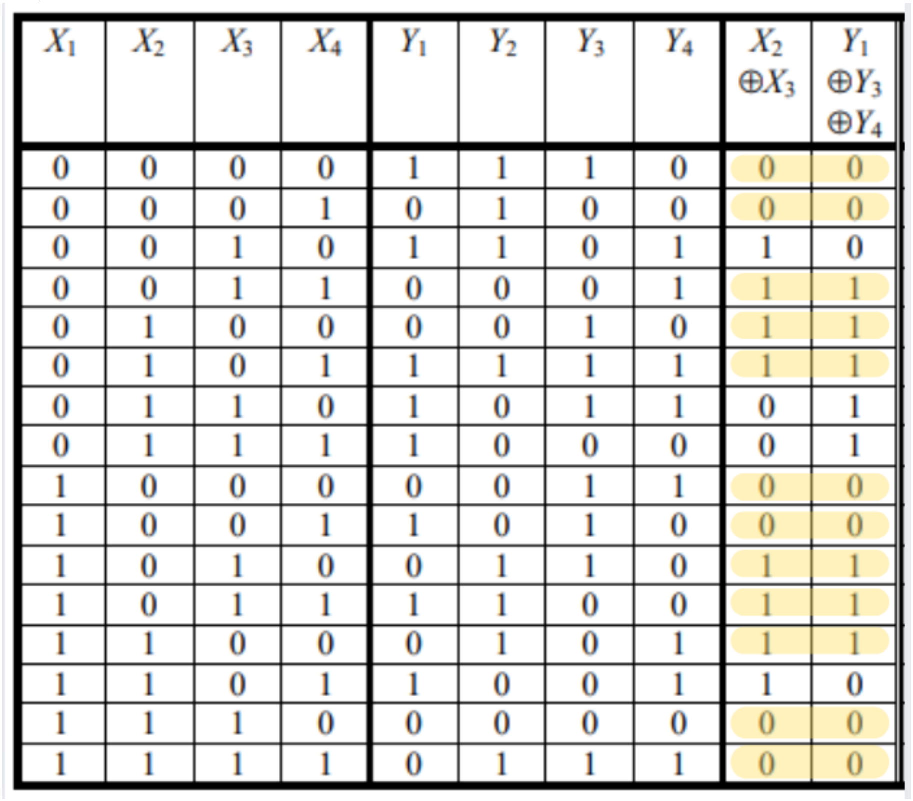


Fig: S-box Mappings

Applying all 16 possible input values for X and examining the corresponding output values Y, it may be observed that for exactly 12 out of the 16 cases, the expression above holds true. Hence, the probability bias is 12/16−1/2 = 1/4. This can be easily seen in the table below.



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

Table: Approximation Table for sample S-box

3.3 Linear Approximation for Basic SPN Cipher

Now that we understand the tools used, we can build and understand the Linear Approximation Table for basic SPN cipher. The linear approximation table below lists all possible linear approximations of the S-box in our cipher. Each value in the table shows how many times a specific linear equation matches the output, adjusted by subtracting 8. To calculate the probability bias for a particular input-output linear combination, divide the table value by 16.

Here's how the values are represented:

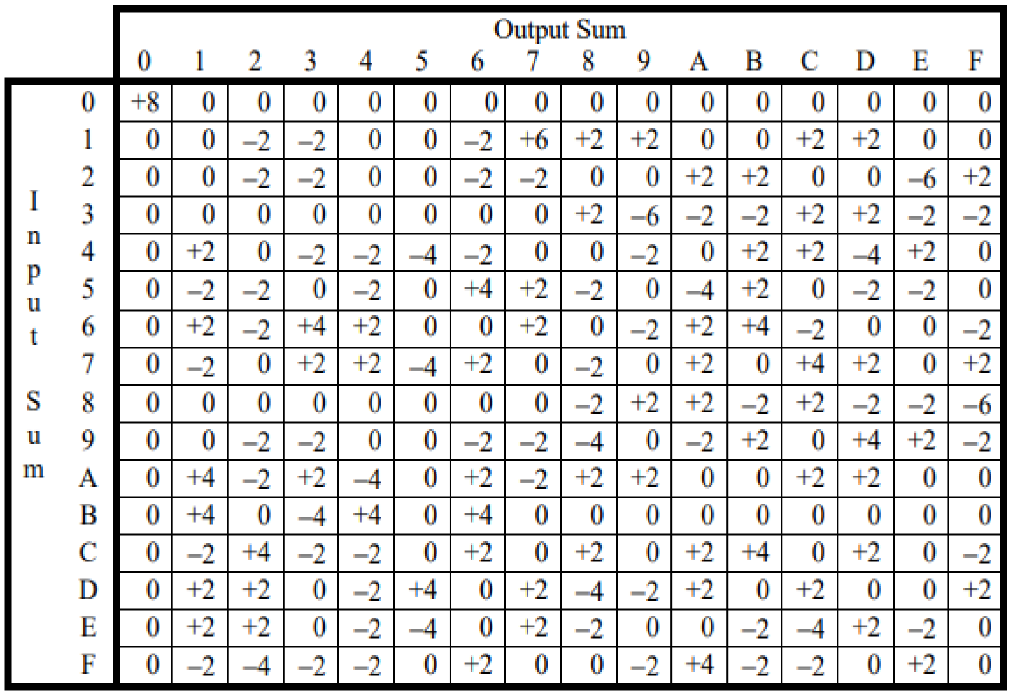
- The **input sum** (hexadecimal) corresponds to a linear combination of input bits, written as a1·X1⊕a2 ·X2⊕a3·X3⊕a4·X4, where ai∈{0,1} and "·" is the binary AND operation. The hexadecimal value is just the binary string a1 a2 a3 a4, where a1 is the most significant bit.

- The **output sum** (hexadecimal) represents a combination of output bits b1·Y1⊕b2·Y2⊕b3·Y3⊕b4·Y4 , where bi∈{0,1}. Similarly, the hexadecimal value is the binary string b1 b2 b3 b4.

For example, consider the linear equation $X3 \oplus X4 = Y1 \oplus Y4$. This is represented by hex input 3 (binary 0011) and hex output 9 (binary 1001). If the table value for this combination is -6, the bias is $-6/16 = -3/8$. The probability that the equation holds is:

$$-3/8 \ + \tfrac{1}{2} = 1/8.$$

This simplifies understanding of how biases and probabilities are calculated for S-box approximations.



Input Sum	Output Sum															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	–2	–2	0	0	–2	+6	+2	+2	0	0	+2	+2	0	0
2	0	0	–2	–2	0	0	–2	–2	0	0	+2	+2	0	0	–6	+2
3	0	0	0	0	0	0	0	0	+2	–6	–2	–2	+2	+2	–2	–2
4	0	+2	0	–2	–2	–4	–2	0	0	–2	0	+2	+2	–4	+2	0
5	0	–2	–2	0	–2	0	+4	+2	–2	0	–4	+2	0	–2	–2	0
6	0	+2	–2	+4	+2	0	0	+2	0	–2	+2	+4	–2	0	0	–2
7	0	–2	0	+2	+2	–4	+2	0	–2	0	+2	0	+4	+2	0	+2
8	0	0	0	0	0	0	0	0	–2	+2	+2	–2	+2	–2	–2	–6
9	0	0	–2	–2	0	0	–2	–2	–4	0	–2	+2	0	+4	+2	–2
A	0	+4	–2	+2	–4	0	+2	–2	+2	+2	0	0	+2	+2	0	0
B	0	+4	0	–4	+4	0	+4	0	0	0	0	0	0	0	0	0
C	0	–2	+4	–2	–2	0	+2	0	+2	0	+2	+4	0	+2	0	–2
D	0	+2	+2	0	–2	+4	0	+2	–4	–2	+2	0	+2	0	0	+2
E	0	+2	+2	0	–2	–4	0	+2	–2	0	0	–2	–4	+2	–2	0
F	0	–2	–4	–2	–2	0	+2	0	0	–2	+4	–2	–2	0	+2	0

Table: Approximation Table for SPN S-box

3.4 Constructing Linear Approximation for complete cipher

3.4.1 What We Are Trying to Achieve

The goal is to construct a **linear approximation** of the cipher over the first three rounds that relate to the following:

- The **plaintext bits** P5, P7, P8,
- Certain intermediate **S-box outputs**, and
- Some unknown **key bits**.

This linear approximation provides a biased probability (not exactly 1/2) that helps us identify relationships between plaintext and key bits.

3.4.2 S-box Approximations

Once the linear approximation information has been compiled for the S-boxes in an SPN, we have the data to determine linear approximations of the overall cipher. This can be achieved by concatenating appropriate linear approximations of S-boxes. Constructing a linear approximation involving plaintext bits and data bits from the output of the second last round of S-boxes makes it possible to attack the cipher by recovering a subset of the subkey bits that follow the last round. We illustrate with an example. Consider an approximation involving S12, S22, S32, and S34, as illustrated below. Note that this develops an expression for the first three rounds of the cipher and not the complete four rounds. We use the following approximations of the S-box:

S12: X1⊕X3⊕X4 = Y2 with probability 12/16 and bias +1/4
S22: X2 = Y2⊕Y4 with probability 4/16 and bias −1/4
S32: X2 = Y2⊕Y4 with probability 4/16 and bias −1/4
S34: X2 = Y2⊕Y4 with probability 4/16 and bias −1/4

3.4.3 Actual Construction

General terms used:

- Ui,j = jth input bit to s-box in round i
- Vi,j = jth output bit from s-box in round i
- Pi = ith bit of plaintext input
- Ki,j = jth bit of key in round i

Construct Linear Approximation for Round 1

In the first round, the linear relationship for V1,6 (the 6th output bit from an S-box in Round 1) is given as: V1,6 = P5 ⊕ P7 ⊕ P8 ⊕ K1,5 ⊕ K1,7 ⊕ K1,8
This approximation holds with a **bias of +1/4** (probability 12/16 = 3/4), which can be verified using the linear approximation table: input sum = 1011 and output sum = 0100.

Extend to Round 2 Using Piling-Up Lemma

In Round 2, the approximation for V2,6 (an intermediate S-box output) is: V2,6 ⊕ V2,8 = U2,6
This equation also has a **bias of -1/4,** which can be verified using the linear approximation table where input sum = 0100 and output sum = 0101.
The output U2,6 of Round 2 depends on V1,6, so we combine their biases:

- Bias of V1,6: +1/4,
- Bias of V2,6: -1/4,
- Combined bias: 2×(1/4)×(-1/4)=-1/8.

Thus, the approximation holds with a bias of −1/8.

Incorporate Round 3

In Round 3, we combine more S-box outputs: V3,6 ⊕ V3,8 = U3,6 and V3,14 ⊕ V3,16 = U3,14
They both hold a **bias of -1/4,** with input sum = 0100 and output sum = 0101
These outputs depend on the inputs from Round 2 and key bits of Round 3. Using the Piling-Up Lemma:

- Bias from Round 2: −1/8,

- Bias of additional S-boxes in Round 3: $-1/4$ and -1/4,
- Combined bias: (2×2)×(-1/8)×(-1/4)×(-1/4)=-1/32.

This bias reflects the reliability of the combined approximation across all three rounds.

Combine All Three Rounds

Now, we combine the results of Rounds 1, 2, and 3 into a single equation that involves:

- Plaintext bits P5, P7, P8,
- Key bits K1,5, K1,7, K1,8, K2,6, ... let be ΣK which will be fixed,
- Intermediate S-box outputs U4,6, U4,8, U4,14, U4,16.

The final equation is:

$$U4,6 \oplus U4,8 \oplus U4,14 \oplus U4,16 \oplus P5 \oplus P7 \oplus P8 = 0$$

This holds with a bias of -1/32, meaning the probability is 15/32 or 17/32, depending on the value of ΣK (the XOR of the key bits) = 0 or 1, respectively.

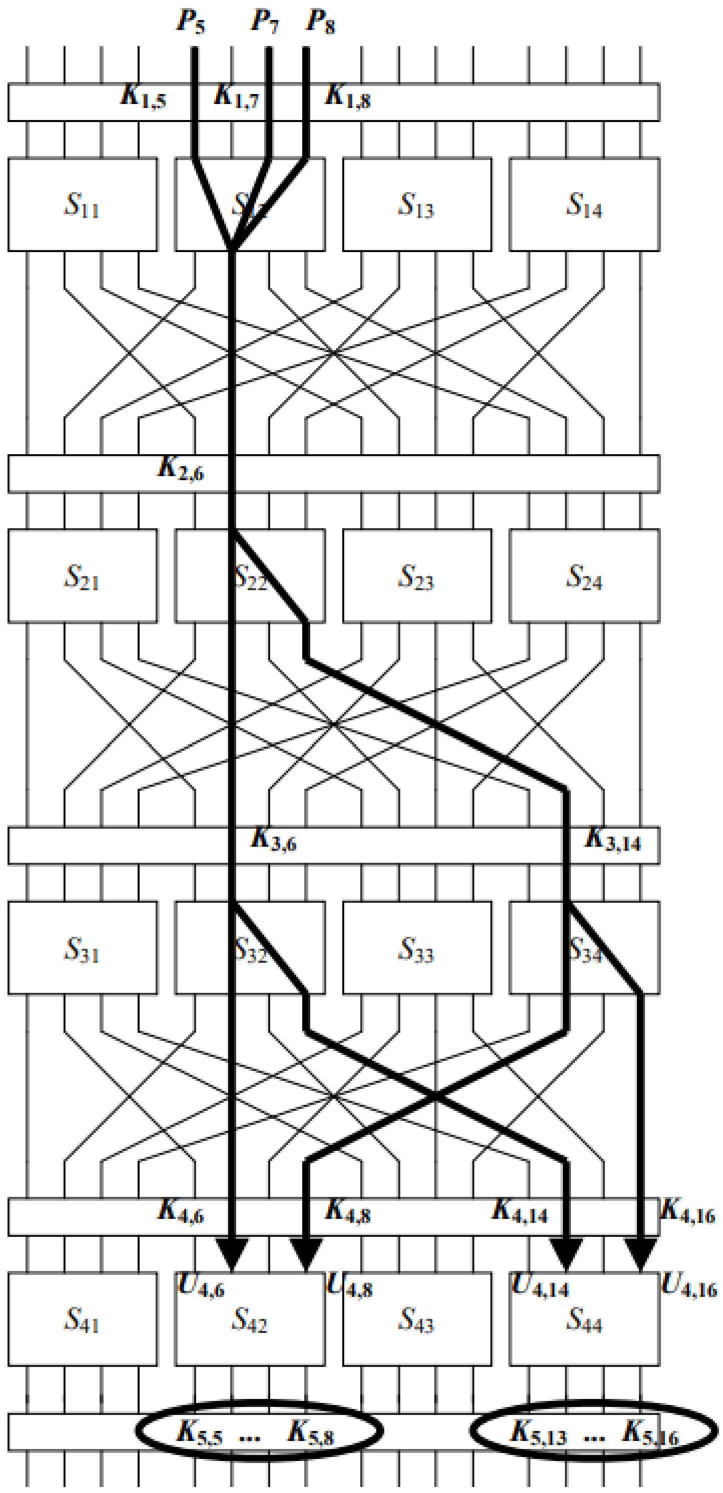


Fig: Construction of Linear Approximation for basic SPN cipher

3.5 Extracting Key bits K5,5-8 and K5,13-16

When performing linear cryptanalysis on a cipher, the goal is to exploit a discovered $\mathrm{R}-1$ round linear approximation (a relationship between plaintext and ciphertext bits) to recover part of the last subkey. This portion of the subkey is called the **target partial subkey** and includes the bits from the last subkey, i.e. [K5,5 ... K5,8] and [K5,13 ... K5,16] that affect the S-boxes involved in the linear approximation.

Here's how the process works:

1. **Partially Decrypt the Last Round**:
   For each possible value of the target partial subkey, we take the ciphertext bits, XOR them with the subkey bits, and reverse them through the relevant S-boxes in the last round. This gives us intermediate values for those data bits.

2. **Count Matches**:
   Using all known plaintext-ciphertext pairs, where at least 8000 pairs are required and we are using 10,000 pairs, we test if the linear approximation holds true with the partially decrypted data. Each time the approximation holds, we increase a counter for the current subkey value.

3. **Identify the Correct Subkey**:
   The correct subkey value is the one whose count differs the most from half the number of plaintext-ciphertext pairs. This works because:

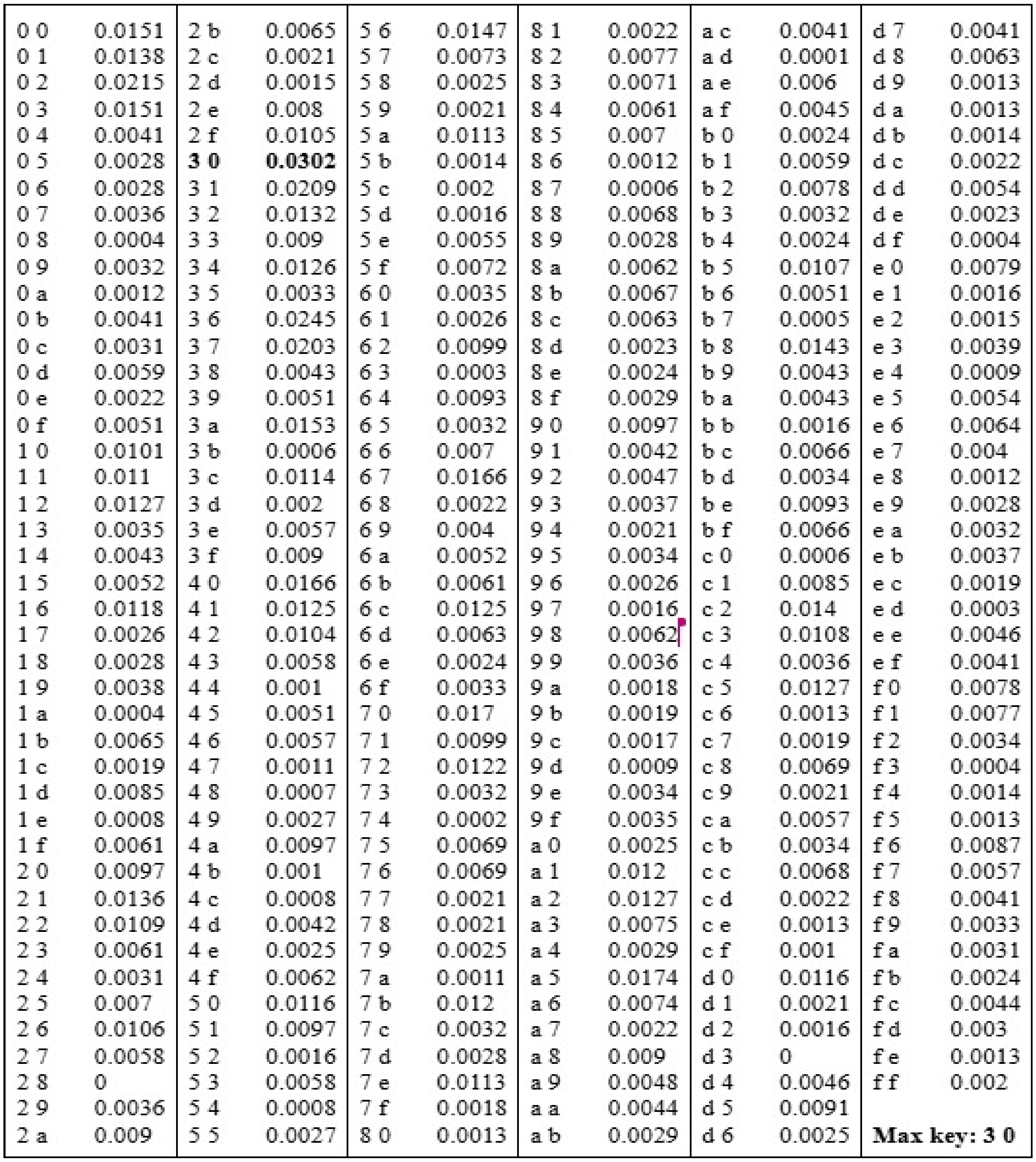
   - If the correct subkey is used, the linear approximation will have a bias (a probability significantly above or below 1/2).
   - If an incorrect subkey is used, the approximation will hold randomly, resulting in a probability close to $\frac{1}{2}$, making absolute bias smaller.

4. **Determine Deviation**:
   The deviation of the count from half of the total samples indicates the strength of the bias. The subkey value with the largest deviation is assumed to be correct.

3.5.1 Pseudo Code

```
C/C++
Initialize Count[L1, L2] for all (L1, L2) ∈ (0,0) to (F,F) as 0

For each (x, y) in T:  // T is the set of plaintext-ciphertext pairs

      For (L1, L2) from (0,0) to (F,F):

            v4<2 NIBBLE> ← L1 ⊕ y<2 NIBBLE>

            v4<4 NIBBLE> ← L2 ⊕ y<4 NIBBLE>

            // $\pi S^{-1}$ is the inverse substitution function

            u4<2 NIBBLE> ← $\pi S^{-1}$(v4<2 NIBBLE>)

            u4<4 NIBBLE> ← $\pi S^{-1}$(v4<4 NIBBLE>)

            z ← x5 ⊕ x7 ⊕ x8 ⊕ u4<6 BIT> ⊕ u4<14 BIT> ⊕ u4<16 BIT>

            If z = 0:

                  Count[L1, L2] ← Count[L1, L2] + 1

max ← -1

For (L1, L2) from (0,0) to (F,F):

      Count[L1, L2] ← (Count[L1, L2] - n/2)/n // n is the length of T

      If Count[L1, L2] > max:

            max ← Count[L1, L2]

            maxkey ← (L1, L2)

Output (maxkey)
```

3.5.2 Example of bias value and maxkey with masterkey = FFFFFFFFFFFF8f2383f0



0 0	0.0151	2 b	0.0065	5 6	0.0147	8 1	0.0022	a c	0.0041	d 7	0.0041
0 1	0.0138	2 c	0.0021	5 7	0.0073	8 2	0.0077	a d	0.0001	d 8	0.0063
0 2	0.0215	2 d	0.0015	5 8	0.0025	8 3	0.0071	a e	0.006	d 9	0.0013
0 3	0.0151	2 e	0.008	5 9	0.0021	8 4	0.0061	a f	0.0045	d a	0.0013
0 4	0.0041	2 f	0.0105	5 a	0.0113	8 5	0.007	b 0	0.0024	d b	0.0014
0 5	0.0028	**3 0**	**0.0302**	5 b	0.0014	8 6	0.0012	b 1	0.0059	d c	0.0022
0 6	0.0028	3 1	0.0209	5 c	0.002	8 7	0.0006	b 2	0.0078	d d	0.0054
0 7	0.0036	3 2	0.0132	5 d	0.0016	8 8	0.0068	b 3	0.0032	d e	0.0023
0 8	0.0004	3 3	0.009	5 e	0.0055	8 9	0.0028	b 4	0.0024	d f	0.0004
0 9	0.0032	3 4	0.0126	5 f	0.0072	8 a	0.0062	b 5	0.0107	e 0	0.0079
0 a	0.0012	3 5	0.0033	6 0	0.0035	8 b	0.0067	b 6	0.0051	e 1	0.0016
0 b	0.0041	3 6	0.0245	6 1	0.0026	8 c	0.0063	b 7	0.0005	e 2	0.0015
0 c	0.0031	3 7	0.0203	6 2	0.0099	8 d	0.0023	b 8	0.0143	e 3	0.0039
0 d	0.0059	3 8	0.0043	6 3	0.0003	8 e	0.0024	b 9	0.0043	e 4	0.0009
0 e	0.0022	3 9	0.0051	6 4	0.0093	8 f	0.0029	b a	0.0043	e 5	0.0054
0 f	0.0051	3 a	0.0153	6 5	0.0032	9 0	0.0097	b b	0.0016	e 6	0.0064
1 0	0.0101	3 b	0.0006	6 6	0.007	9 1	0.0042	b c	0.0066	e 7	0.004
1 1	0.011	3 c	0.0114	6 7	0.0166	9 2	0.0047	b d	0.0034	e 8	0.0012
1 2	0.0127	3 d	0.002	6 8	0.0022	9 3	0.0037	b e	0.0093	e 9	0.0028
1 3	0.0035	3 e	0.0057	6 9	0.004	9 4	0.0021	b f	0.0066	e a	0.0032
1 4	0.0043	3 f	0.009	6 a	0.0052	9 5	0.0034	c 0	0.0006	e b	0.0037
1 5	0.0052	4 0	0.0166	6 b	0.0061	9 6	0.0026	c 1	0.0085	e c	0.0019
1 6	0.0118	4 1	0.0125	6 c	0.0125	9 7	0.0016	c 2	0.014	e d	0.0003
1 7	0.0026	4 2	0.0104	6 d	0.0063	9 8	0.0062	c 3	0.0108	e e	0.0046
1 8	0.0028	4 3	0.0058	6 e	0.0024	9 9	0.0036	c 4	0.0036	e f	0.0041
1 9	0.0038	4 4	0.001	6 f	0.0033	9 a	0.0018	c 5	0.0127	f 0	0.0078
1 a	0.0004	4 5	0.0051	7 0	0.017	9 b	0.0019	c 6	0.0013	f 1	0.0077
1 b	0.0065	4 6	0.0057	7 1	0.0099	9 c	0.0017	c 7	0.0019	f 2	0.0034
1 c	0.0019	4 7	0.0011	7 2	0.0122	9 d	0.0009	c 8	0.0069	f 3	0.0004
1 d	0.0085	4 8	0.0007	7 3	0.0032	9 e	0.0034	c 9	0.0021	f 4	0.0014
1 e	0.0008	4 9	0.0027	7 4	0.0002	9 f	0.0035	c a	0.0057	f 5	0.0013
1 f	0.0061	4 a	0.0097	7 5	0.0069	a 0	0.0025	c b	0.0034	f 6	0.0087
2 0	0.0097	4 b	0.001	7 6	0.0069	a 1	0.012	c c	0.0068	f 7	0.0057
2 1	0.0136	4 c	0.0008	7 7	0.0021	a 2	0.0127	c d	0.0022	f 8	0.0041
2 2	0.0109	4 d	0.0042	7 8	0.0021	a 3	0.0075	c e	0.0013	f 9	0.0033
2 3	0.0061	4 e	0.0025	7 9	0.0025	a 4	0.0029	c f	0.001	f a	0.0031
2 4	0.0031	4 f	0.0062	7 a	0.0011	a 5	0.0174	d 0	0.0116	f b	0.0024
2 5	0.007	5 0	0.0116	7 b	0.012	a 6	0.0074	d 1	0.0021	f c	0.0044
2 6	0.0106	5 1	0.0097	7 c	0.0032	a 7	0.0022	d 2	0.0016	f d	0.003
2 7	0.0058	5 2	0.0016	7 d	0.0028	a 8	0.009	d 3	0	f e	0.0013
2 8	0	5 3	0.0058	7 e	0.0113	a 9	0.0048	d 4	0.0046	f f	0.002
2 9	0.0036	5 4	0.0008	7 f	0.0018	a a	0.0044	d 5	0.0091		
2 a	0.009	5 5	0.0027	8 0	0.0013	a b	0.0029	d 6	0.0025	**Max key: 3 0**	

3.6 Full key recovery

Linear cryptanalysis is a *known plaintext attack*, which means the adversary has access to a set of plaintexts and their corresponding ciphertexts generated by the encryption algorithm under the secret key. In addition to the plaintext-ciphertext pairs, the adversary also has access to an *encryption oracle.* An encryption oracle is essentially a black-box system that takes any plaintext input and returns its corresponding ciphertext using the same encryption key. This allows the adversary to query the oracle for the ciphertext of any plaintext they choose without knowing the encryption key.

In our scenario, the adversary first performs a linear cryptanalysis to find part of the key. Specifically, the attack exploits statistical biases in linear approximations of the cipher's S-boxes to recover **8 bits,** that is [K5,5 ... K5,8] and [K5,13 ... K5,16] the last round's subkey or is bits 69-72 and 77-80 of the master key. These bits, which we got from the analysis of the linear equation and the plaintext-ciphertext pairs, serve as a starting point for further key recovery.

3.6.1 Exhaustive Search for the Remaining Key Bits

After obtaining these 8 bits, the adversary still needs to recover the remaining **72 bits** of the 80-bit key. This is done using an *exhaustive search*, also known as a brute-force attack. Here's the process:

1. **Combining Information from Linear Cryptanalysis**:
   The 8 bits obtained from the linear attack reduce the search space of the key significantly. Instead of brute-forcing all 80 bits of the key (which would involve 2^{80} combinations), the adversary only needs to brute-force the remaining 2^{72} combinations, making the attack computationally feasible.

2. **Testing Candidate Keys Using the Encryption Oracle**:
   The adversary constructs a candidate key for each possible value of the 72 unknown bits (while keeping the eight known bits fixed). Using this candidate key:

   - The adversary encrypts the plaintexts from their known plaintext-ciphertext pairs through the encryption oracle.
   - They then compare the resulting ciphertexts to the actual ciphertexts in their possession.

3. **Identifying the Correct Key**:
   Since encryption is deterministic, the correct key will produce ciphertexts that perfectly match the known ciphertexts for all plaintext inputs. Therefore, the adversary can identify the correct key by testing all candidate keys until a perfect match is found.

4. **Conclusion**

This report outlines the critical aspects of linear cryptanalysis as applied to Substitution-Permutation Network (SPN) ciphers, emphasising its practical implications for modern cryptography and the vulnerabilities of symmetric-key algorithms. We began by implementing and understanding a basic 4-round SPN cipher, providing a foundation for analysing its encryption process. Through linear cryptanalysis, we successfully recovered specific portions of the final round key, demonstrating the effectiveness of this cryptanalytic technique in exploiting statistical biases.

Subsequently, we attempted an exhaustive search to recover the full master key. However, as expected, the time complexity of this brute-force approach proved to be computationally expensive. To mitigate this, we fixed the first 48 bits of the master key to a constant value (F), reducing the search space. This modification allowed us to perform encryption, linear cryptanalysis, and exhaustive search more efficiently for testing and analysis purposes. Through this process, we demonstrated the practicality and effectiveness of linear cryptanalysis as a cryptanalytic technique. By fixing portions of the key, we emphasised the trade-offs between security and computational feasibility, providing insights into how real-world constraints can impact cryptographic implementations as we found the last eight nibbles in approximately 3 to 5 minutes through our code. Then, we can estimate the time required to recover all 20 nibbles (80 bits) by scaling the effort proportionally. Since $80-32=48$, the search space is 2^{48} times larger than that for 32 bits. Assuming the average time to find 32 bits is 4 minutes (the midpoint of 3–5 minutes), the total time to find 80 bits would be $4\times2^{48} = 4\times281{,}474{,}976{,}710{,}656$ minutes = (approximately) 2.14×1015 years.

In conclusion, while linear cryptanalysis proved its mettle by nibbling away at a few bits of the master key with statistical finesse, brute-forcing the entire key turned out to be a cosmic joke—requiring about 2.14 quadrillion years. To put that in perspective, you'd need to start cracking during the Big Bang and still miss the deadline by a few universes. This highlights an essential cryptographic lesson: when it comes to key size, bigger isn't just better—it's the difference between a solvable puzzle and a celestial pipe dream. So, remember, friends: if you're picking a key, think big, or risk having your cipher cracked in the next trillion millennia!

References

- A Tutorial on Linear and Differential Cryptanalysis by Howard M. Heys
- Book - Cryptography theory and practice by Stinson and Paterson
- Substitution - permutation network by Wikipedia
- Understanding Cryptography: A Textbook for Students and Practitioners by Cristof Parr and Jen Pelzl