Mischief in the Cube: Study of Saturnin

Term Paper submitted to
Indian Institute of Technology, Bhilai

by

Kriti Arora

Under the guidance of

Dr. Dhiman Saha

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY BHILAI

Contents

Abstract

This term paper presents a comprehensive study of lightweight cryptography, focusing on the Saturnin cipher. We explore the design principles of lightweight ciphers and their importance in modern constrained environments. A key contribution of this work is the development and analysis of a "toy" version of Saturnin, designed to facilitate understanding of lightweight cipher design from an undergraduate perspective. This study aims to bridge the gap between theoretical cipher specifications and practical understanding through implementation and cryptanalysis.

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Lightweight Cryptography

Lightweight cryptography is a subfield of cryptography that aims to provide security for devices with limited resources, such as RFID tags, sensor networks, and embedded systems. Unlike conventional cryptography, which prioritizes high security and performance on powerful platforms, lightweight cryptography balances security, performance, and implementation cost (area, power, energy).

1.2 Saturnin Specifications

Saturnin is a block cipher designed for post-quantum security. It is an SPN (Substitution-Permutation Network) cipher that operates on 256-bit blocks and supports a 256-bit key. Its design is optimized for both hardware and software implementations, aiming to be secure against quantum attacks while remaining lightweight.

1.3 Study Summary

This thesis presents a comprehensive study of the Saturnin block cipher, focusing on its cryptanalytic properties and hardware efficiency. We developed a toy version of Saturnin to facilitate educational understanding and experimental verification of various attacks. Our research encompasses a wide range of cryptanalytic techniques, including differential, lin-

ear, differential-linear, boomerang, and higher-order differential analysis. We also employed automated tools like MILP and SAT solvers to find optimal trails and verify properties. Furthermore, we implemented the cipher's round function in Verilog to evaluate its hardware complexity and performance, comparing it with other lightweight ciphers.

1.4 Contributions and Findings

The key contributions and findings of this research are:

- **Toy Cipher Development:** Created a scaled-down "Toy Saturnin" to demonstrate attacks and properties practically.
- **Cryptanalysis:**
  - Verified the impossibility of Bogi permutation for Saturnin's structure.
  - Identified impossible differential trails.
  - Constructed and verified Differential-Linear distinguishers.
  - Confirmed the algebraic degree and zero-sum properties for up to 2 rounds.
  - Performed Boomerang analysis and identified incompatibilities in the S-box.
- **Hardware Implementation:**
  - Implemented a flattened one-round Saturnin module in Verilog.
  - Derived Algebraic Normal Form (ANF) equations for the S-box.
  - Demonstrated that Saturnin's S-box has a comparable area (approx. 25 GE) to other lightweight ciphers while offering maximum algebraic degree (3).
- **Automated Analysis:**
  - Modeled the differential propagation using MILP, finding an optimal branch number of 2.0 for the S-box and a full round differential trail weight of 4.0.
  - Generated 222 convex hull inequalities to describe the S-box's differential behavior.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows:

- **Chapter 2: Background** covers the original cipher introduction, the toy cipher explanation, notations, and an explanation of cryptanalysis techniques.
- **Chapter 3: Cryptanalysis** details the application of various cryptanalytic techniques including Bogi, Impossible Differential, Linear, Differential-Linear, Higher Order Differential, Zero Sum Distinguishers, Truncated, and Boomerang attacks.
- **Chapter 5: Hardware Implementation** presents the one-round Verilog implementation, a comparative hardware study, and observations from Cadence results.
- **Chapter 6: Automated Tools** discusses the 1-round MILP model and the 1-round SAT solver.
- **Chapter 7: Discussion** explores why Saturnin is considered lightweight and the philosophy behind its design.
- **Chapter 8: Conclusion** summarizes the work and discusses future directions.
- **Appendix** contains additional material and code listings.

CHAPTER 2

Background

2.1 Original Cipher Introduction

Saturnin is a block cipher designed to be secure even against quantum computers (post-quantum security). It is a Substitution-Permutation Network (SPN) that uses a 256-bit key and a 256-bit block size.

2.1.1 State Representation

The internal state of Saturnin is 256 bits. It is represented as a $4 \times 4 \times 4$ cube of nibbles (4-bit units). This means there are 64 nibbles in total. Alternatively, the state can be viewed as 16 registers, where each register has 16 bits.

2.1.2 S-boxes

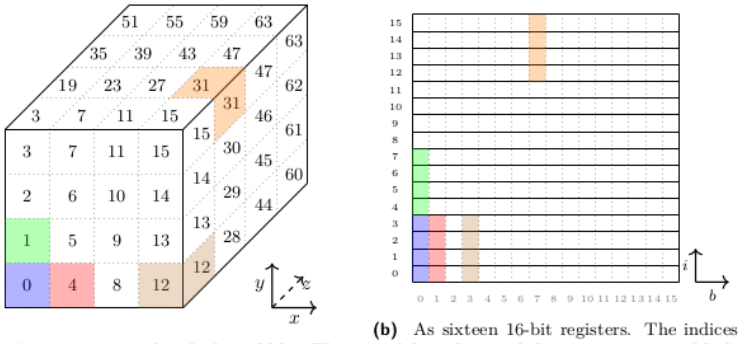
Saturnin uses two different S-boxes, S_0 and S_1, which are 4-bit non-linear permutations.

- S_0 is applied to nibbles with even indices.
- S_1 is applied to nibbles with odd indices.

The lookup tables for the S-boxes are:

2.1.3 ShiftRows Operations

Saturnin uses two types of permutations to mix the nibbles:



(a) As a $4 \times 4 \times 4$ cube of 4-bit nibbles. The boundaries between the nibbles are in gray.

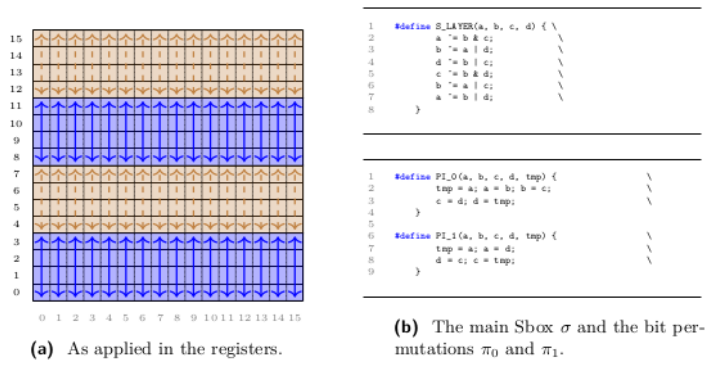
(b) As sixteen 16-bit registers. The indices and boundaries of the registers are in black, those of the bits are in gray.

Figure 1: The two representations of the 256-bit state of SATURNIN. Nibbles and their corresponding bits are represented with the same color in each representation.

Figure 2.1: MDS Matrix (Source: Saturnin Spec)

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_0(x)$	0	6	14	1	15	4	7	13	9	8	12	5	2	10	3	11
$S_1(x)$	0	9	13	2	15	1	11	7	6	4	5	3	8	12	10	14

Table 2.1: Saturnin S-boxes



(a) As applied in the registers.

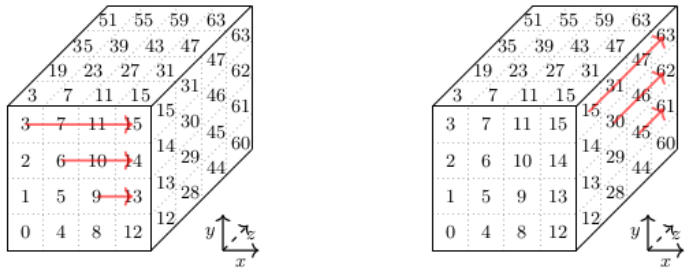
(b) The main Sbox σ and the bit permutations π_0 and π_1.

Figure 11: The application of the 4-bit Sboxes (S) in the register representation. σ_0 is represented by continuous blue arrows, σ_1 by dashed brown ones.

Figure 2.2: S-box implementation (Source: Saturnin Spec)

- **SR-slice**: This operation works on "slices" of the cube. It moves the nibbles within a slice. Specifically, it maps a nibble at position (x, y, z) to $(x + y \pmod 4), y, z)$.
- **SR-sheet**: This operation works on "sheets". It moves nibbles within a sheet. It maps a nibble at position (x, y, z) to $(x, y, z + y$

(mod 4)).



(a) $\mathsf{SR}_{\text{slice}}$ (when $r \equiv 1 \bmod 4$) **(b)** $\mathsf{SR}_{\text{sheet}}$ (when $r \equiv 3 \bmod 4$)

Figure 5: Representation of the SR_r operations on the cube.

Figure 2.3: ShiftRows Operation (Source: Saturnin Spec)

2.1.4 MDS Matrix

The linear layer uses an MDS (Maximum Distance Separable) matrix. This matrix mixes the columns of the state. The branch number of this matrix is 5, which means it provides excellent diffusion (spreading of changes). If you change one input nibble, it affects all output nibbles in that column.

2.1.5 Super-Round Algorithm

Saturnin organizes its rounds into "super-rounds". One super-round consists of two consecutive rounds (an even round and an odd round).

- **Even Round**:
  1. Apply S-boxes (S_0, S_1).
  2. Apply MDS matrix.
- **Odd Round**:
  1. Apply S-boxes (S_0, S_1).
  2. Apply ShiftRows (either SR-slice or SR-sheet depending on the round number).
  3. Apply MDS matrix.
  4. Apply Inverse ShiftRows.
  5. Add Round Constant and Key.

2.2 Toy Cipher Explanation

The original Saturnin cipher operates on a 16×16 state, i.e., a matrix of 16 rows and 16 columns of 4-bit cells, totaling 256 bits. In contrast, our toy variant significantly reduces the state size for analysis and experimentation.

Toy Cipher State. The toy design uses an 8×4 state, consisting of:

$$8 \text{ registers} \times 4 \text{ bits each} = 32 \text{ bits}.$$

These registers are denoted $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$, where each x_i is a 4-bit value.

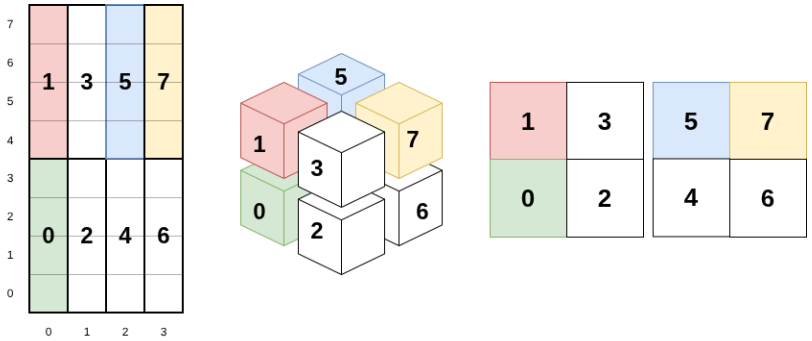


Figure 2.4: Toy Cipher Structure

2.2.1 Nonlinear Layer: Bitsliced S-box Application

The cipher state consists of eight 4-bit registers:

$$S = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7), \quad x_i \in \{0, \ldots, 15\}.$$

Bitslice Structure

The eight registers are divided into two groups:

$$G_0 = (x_0, x_1, x_2, x_3), \quad G_1 = (x_4, x_5, x_6, x_7).$$

Each group contains four 4-bit words. For bit position $j \in \{0, 1, 2, 3\}$, we define the slice

$$(a_j, b_j, c_j, d_j) = ((u_0)_j, (u_1)_j, (u_2)_j, (u_3)_j).$$

Thus, the design applies **8 S-boxes in parallel** — four from G_0 and four from G_1.

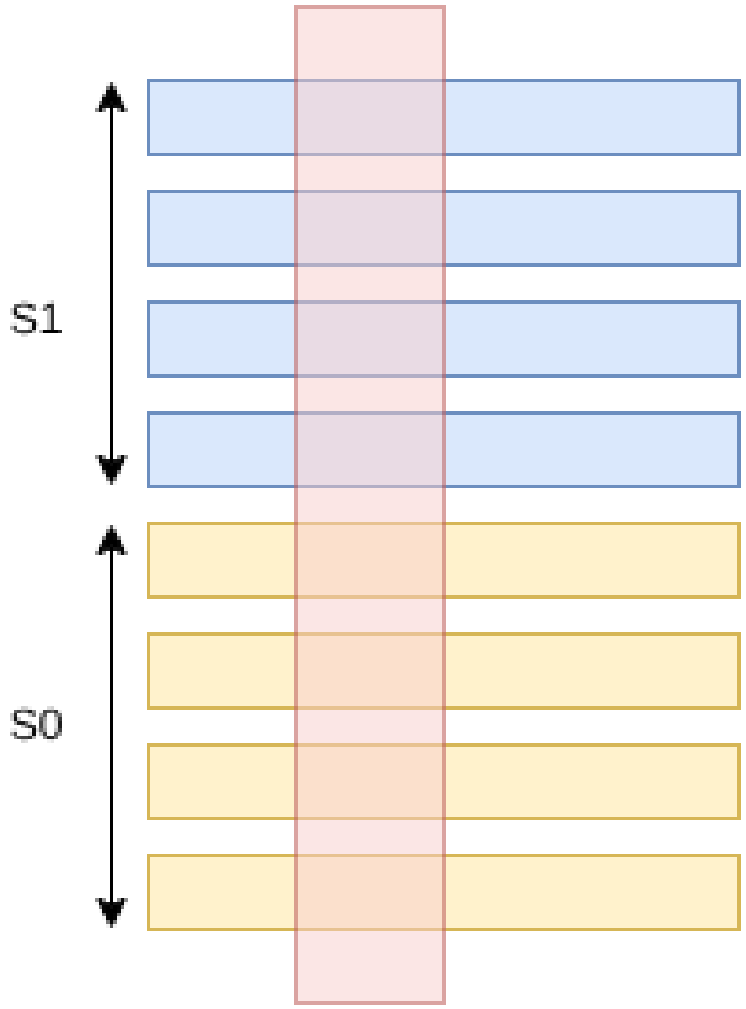


Figure 2.5: Bitsliced S-box Application

Group-specific S-box Variants

- S_0 is applied to all slices of G_0.
- S_1 is applied to all slices of G_1.

2.2.2 Shift Operations: SR_slice and SR_sheet

SR_slice

In this operation, the top row is shifted left, while the bottom row remains unchanged.

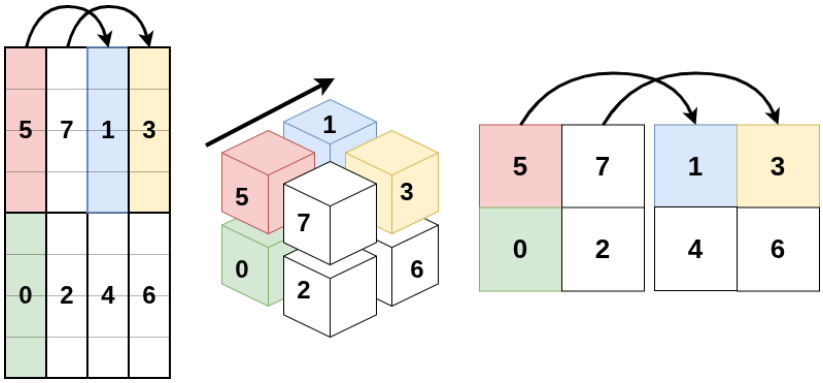


Figure 2.6: Toy SR-slice Operation

SR_sheet

In this operation, the top row is shifted, while the bottom row remains unchanged.

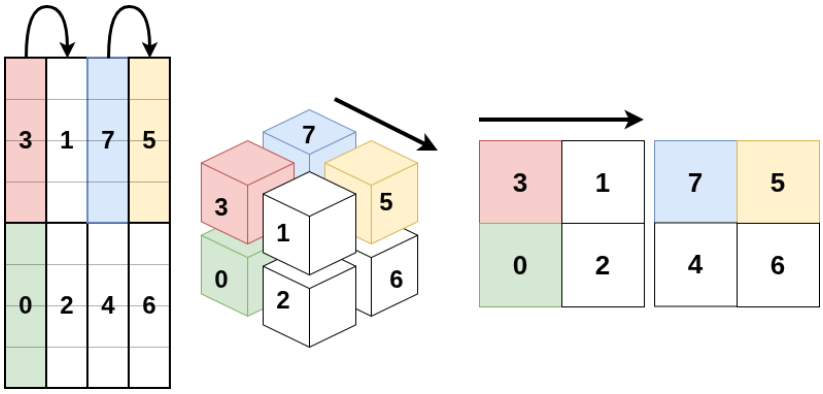


Figure 2.7: Toy SR-sheet Operation

2.2.3 MDS Layer

Original Saturnin MDS

Each group (t_0, t_1, t_2, t_3) undergoes:

$$(t_0, t_1, t_2, t_3) \leftarrow (t_1, t_2, t_3, t_0 \oplus t_1).$$

Toy Cipher MDS

The MDS matrix used in the Toy Cipher is not a full Maximum Distance Separable matrix in the strict sense. It is a scaled-down version derived

from the original Saturnin MDS to provide diffusion for the smaller state size. The branch number of this matrix is approximately 4.5 - 5.

$$(t_0, t_1) \leftarrow (t_1, t_0 \oplus t_1).$$

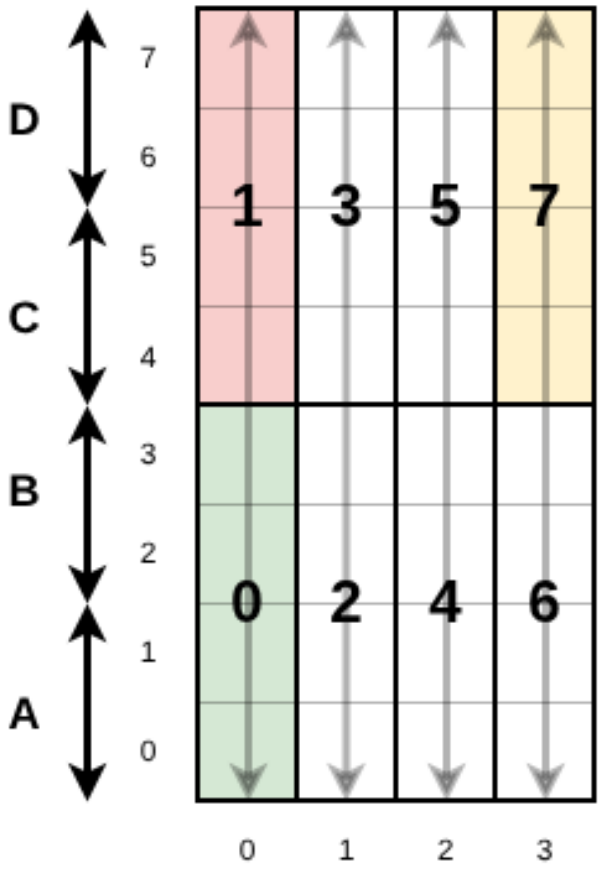


Figure 2.8: Toy Cipher MDS Matrix

2.2.4 Empirical Properties of the MDS Layer

Correctness of the MDS and Its Inverse

For input $(0, 0, 0, 1, 0, 0, 0, 0)$, MDS produced $(0, 1, 1, 0, 0, 1, 0, 0)$, and inverse MDS recovered the original input.

Diffusion Measurement

Measured average diffusion:

3.38 output nibbles changed per input bit flip.

This indicates moderate diffusion, spreading each bit flip across multiple registers.

CHAPTER 3

Cryptanalysis

3.1 Bogi

Bogi is a cryptanalytic technique that typically involves analyzing the structural properties of the cipher to identify distinguishers or weaknesses that deviate from random behavior. It is often used in the context of specific lightweight ciphers to find integral or differential properties.

in\out	1	2	4	8
1	2	0	0	4
2	4	0	0	2
4	0	4	0	0
8	0	2	4	0

Table 3.1: σ_0 1→1 DDT

in\out	1	2	4	8
1	0	2	4	0
2	0	4	2	0
4	0	0	0	4
8	4	0	0	2

Table 3.2: σ_1 1→1 DDT

All inputs are bad inputs and all outputs are bad outputs hence a bogi permutation using the PRESENT's SPN structure is not possible.

3.2 Impossible Differential

Impossible Differential Cryptanalysis exploits the existence of differentials with a probability of exactly zero. By identifying input differences that cannot produce certain output differences, an attacker can discard wrong key guesses that would lead to such impossible events, thereby reducing the key space.

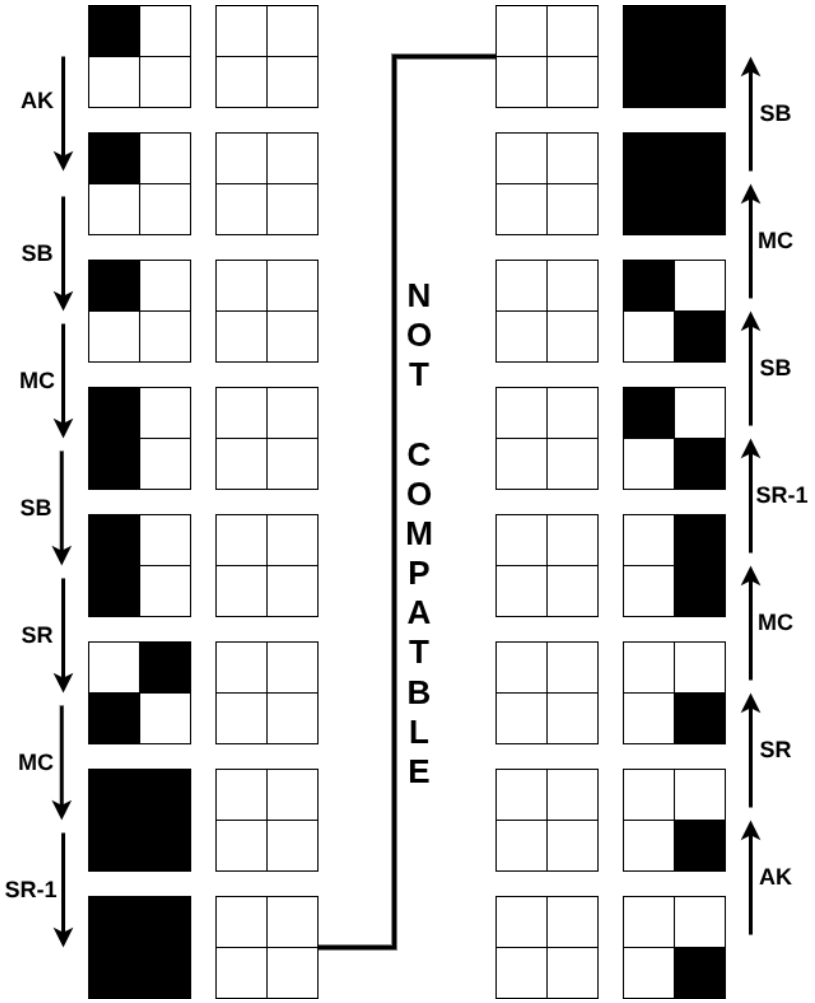


Figure 3.1: Impossible Differential Trail

Similarly, many trails can be found by changing the active nibble value, and key recovery is possible.

3.3 Linear

Linear Cryptanalysis involves approximating the non-linear components of a cipher (like S-boxes) with linear equations. By finding linear approximations that hold with a probability significantly different from 1/2, an attacker can collect plaintext-ciphertext pairs to recover bits of the key.

The Linear Approximation Tables (LAT) for the S-boxes σ_0 and σ_1 are given below. The entries represent the correlation between input and output

linear masks.

in\out	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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	-2	0	0	-2	2	4	4	-2	2
2	0	4	0	0	2	2	2	-2	0	4	0	0	-2	-2	-2	2
3	0	-4	2	-2	2	2	0	0	0	4	2	-2	2	2	0	0
4	0	2	4	-2	0	-2	4	2	0	-2	0	-2	0	2	0	2
5	0	2	2	0	0	-2	-2	0	-4	2	2	4	0	2	2	0
6	0	-2	0	2	2	0	2	4	0	2	-4	2	-2	0	2	0
7	0	-2	2	0	2	0	0	-2	4	-2	2	4	-2	0	0	2
8	0	0	0	0	-4	0	4	0	2	2	2	2	2	-2	2	-2
9	0	0	-2	2	0	4	2	2	-2	-2	4	0	-2	2	0	0
10	0	0	0	4	-2	-2	-2	2	2	2	2	-2	0	0	0	4
11	0	0	2	2	2	2	0	0	-2	-2	0	0	4	-4	2	2
12	0	2	-4	-2	4	-2	0	2	2	0	2	0	2	0	2	0
13	0	2	2	0	0	2	-2	4	2	0	0	2	2	0	-4	-2
14	0	2	0	-2	-2	4	-2	0	2	0	-2	0	0	2	4	2
15	0	2	2	4	2	0	0	-2	2	0	0	-2	0	2	2	-4

Table 3.3: LAT for σ_0

in\out	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	4	0	4	-2	2	2	-2	-2	-2	2	2
2	0	2	4	2	0	-2	4	-2	0	2	0	-2	0	-2	0	2
3	0	2	-4	2	0	2	4	2	2	0	-2	0	2	0	-2	0
4	0	0	2	-2	0	0	-2	2	4	4	-2	2	0	0	-2	2
5	0	0	2	-2	-4	0	2	2	2	-2	0	0	2	2	4	0
6	0	2	-2	0	0	-2	2	0	0	2	2	4	-4	2	2	0
7	0	2	-2	0	4	-2	-2	0	2	0	0	-2	2	0	4	2
8	0	-4	0	0	2	2	2	-2	0	4	0	0	2	2	2	-2
9	0	0	0	4	-2	-2	-2	2	-2	2	2	2	4	0	0	0
10	0	-2	0	-2	2	0	2	0	0	-2	4	2	2	0	-2	4
11	0	2	0	2	-2	4	-2	-4	2	0	2	0	0	2	0	2
12	0	4	2	-2	2	2	0	0	-4	0	-2	2	2	2	0	0
13	0	0	2	2	2	2	0	0	2	-2	0	4	0	-4	2	-2
14	0	-2	2	4	2	0	0	2	0	-2	-2	0	-2	4	0	2
15	0	2	2	0	2	0	0	2	2	0	4	-2	0	2	-2	-4

Table 3.4: LAT for σ_1

The maximum absolute value in the LAT is 4, which indicates the linearity properties of the S-boxes.

3.4 Differential Linear

Differential-Linear Cryptanalysis combines the principles of differential and linear cryptanalysis. It typically splits the cipher into two parts: a differential characteristic is used for the first part, and a linear approximation is used for the second part. This allows the attack to be extended to more rounds than either technique could cover individually.

3.4.1 Experimental Verification

To verify the differential-linear properties, we used a script `dl_distinguisher.py` that tests random input differences and output masks to estimate the bias of the differential-linear approximation. The script generates random plaintexts, applies an input difference, encrypts them for 1 round, and checks the parity of the output mask.

The top biases found from the experiments are:

- $\Delta = [0,0,0,0,0,0,8,0], \Gamma = [0,0,0,0,0,0,3,0]$, bias=1.0000, data$\approx$1
- $\Delta = [0,0,5,0,0,0,0,0], \Gamma = [0,0,4,0,0,0,0,0]$, bias=-0.2568, data$\approx$15
- $\Delta = [0,0,4,0,0,0,0,0], \Gamma = [0,0,6,0,0,0,0,0]$, bias=-0.2468, data$\approx$16

A bias of 1.0 indicates a deterministic relationship, which is a very strong distinguisher. Lower biases require more data to distinguish from random noise. The estimated data complexity is inversely proportional to the square of the bias.

3.4.2 Differential-Linear Connectivity Table (DLCT)

The Differential-Linear Connectivity Table (DLCT) allows us to evaluate the correlation between an input difference and an output linear mask. It is defined as:

$$DLCT(\Delta, \Gamma) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\Gamma \cdot (S(x) \oplus S(x \oplus \Delta))}$$

The DLCT for the S-box is given below:

3.5 Higher Order Differential

Higher Order Differential Cryptanalysis generalizes standard differential cryptanalysis by considering derivatives of higher orders. It exploits the fact that the algebraic degree of the cipher's output bits as a function of the input bits may not be maximal.

3.5.1 Boolean Derivative

The Boolean derivative of a function $f : \{0,1\}^n \to \{0,1\}$ in the direction a is defined as:

$$\Delta_a f(x) = f(x) \oplus f(x \oplus a)$$

This measures whether flipping bits specified by mask a changes the output.

Δ\Γ	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	0	0	0	8	0	8	0	-8	0	0	-8	-8	-8	0	0
2	16	-8	8	-8	0	0	8	0	0	-8	0	0	0	0	0	-8
3	16	0	8	0	8	-8	0	-8	0	0	-8	0	0	0	-8	0
4	16	0	-8	0	0	0	0	-8	-8	8	0	-8	8	8	-8	0
5	16	0	-8	0	0	-8	0	0	8	0	-8	8	-8	-8	0	8
6	16	-8	-8	0	0	0	0	0	0	-8	8	0	0	0	0	0
7	16	0	-8	-8	0	0	0	0	0	0	0	0	0	0	8	-8
8	16	8	0	-8	-8	-8	0	0	0	8	0	8	0	0	-8	-8
9	16	0	0	0	-8	0	0	8	-8	0	0	-8	0	0	0	0
10	16	-8	0	8	-8	0	0	-8	0	-8	-8	0	0	8	0	8
11	16	0	0	0	-8	8	0	0	0	0	0	0	-8	-8	0	0
12	16	8	0	0	0	0	-8	0	-8	0	-8	-8	0	0	8	0
13	16	0	0	-8	0	8	-8	-8	8	0	0	0	0	0	0	-8
14	16	-8	0	0	0	-8	-8	8	0	-8	0	0	8	0	0	0
15	16	0	0	8	0	0	-8	0	0	0	8	0	-8	-8	-8	0

Table 3.5: DLCT for S-box

3.5.2 Higher-Order Differentials and Algebraic Degree

A k-th order differential involves taking derivatives in k distinct directions $a_1, \ldots, a_k$. This corresponds to summing the outputs over an affine cube of dimension k. A crucial property relates this to the algebraic degree of the function:

$$\Delta_{a_1,\ldots,a_k} f(x) = 0 \quad \forall x \iff \deg(f) < k$$

3.6 Zero Sum Distinguishers

Zero Sum Distinguishers are a type of integral property where a set of inputs is chosen such that they sum to zero (usually XOR sum). If the corresponding outputs also sum to zero, this property can be used to distinguish the cipher.

3.6.1 Global Zero-Sum Property

For an n-bit S-box S, the global zero-sum property holds if the XOR sum of all possible outputs is zero:

$$\bigoplus_{x=0}^{2^n-1} S(x) = 0$$

This indicates that the S-box is globally balanced, with outputs evenly distributed.

3.6.2 Zero-Sum Subset (Trail)

A zero-sum trail exists if a nontrivial subset of inputs $\mathcal{S} \subset \{0,1\}^n$ sums to zero:

$$\bigoplus_{x \in \mathcal{S}} S(x) = 0$$

This reveals internal symmetries or structural weaknesses where specific groups of inputs cancel out.

3.6.3 Connection to Higher-Order Differentials

The zero-sum property is practically an integral property. Checking for a zero-sum over a specific set of inputs (like an affine cube) is equivalent to verifying if the higher-order differential vanishes. For instance, a zero-sum over a 4-dimensional cube implies the 4th order derivative is zero, consistent with a function of degree ≤ 3.

3.6.4 Results for Toy Saturnin S-boxes

Experiments show that both S-boxes (S_0 and S_1) in the Toy Saturnin cipher satisfy the global zero-sum property. Additionally, they exhibit nontrivial zero-sum trails, such as the subset of inputs $\{1, 2, 9\}$ for S_0, where $S_0(1) \oplus S_0(2) \oplus S_0(9) = 0$.

3.7 Boomerang

Boomerang Cryptanalysis is an adaptive chosen-plaintext and ciphertext attack that uses two independent differential characteristics for the cipher. It allows an attacker to distinguish the cipher from a random permutation by checking for a specific dependency between quartets of plaintexts and ciphertexts, effectively "returning" a difference through the cipher.

3.7.1 Boomerang Connectivity Table (BCT)

The Boomerang Connectivity Table (BCT) evaluates the probability of the boomerang switch through an S-box. It is defined as:

$$BCT(\Delta, \nabla) = \#\{x \in \mathbb{F}_2^n \mid S^{-1}(S(x) \oplus \nabla) \oplus S^{-1}(S(x \oplus \Delta) \oplus \nabla) = \Delta\}$$

where Δ is the input difference and ∇ is the output difference for the boomerang switch.

3.7.2 Boomerang Difference Table (BDT)

The Boomerang Difference Table (BDT) refines the BCT by taking into account the intermediate difference β of the upper differential characteristic. It is defined as:

$$BDT(\Delta, \beta, \nabla) = \#\{x \in \mathbb{F}_2^n \mid S(x) \oplus S(x \oplus \Delta) = \beta \text{ and } S^{-1}(S(x) \oplus \nabla) \oplus S^{-1}(S(x \oplus \Delta) \oplus \nabla)$$

3.7.3 BCT Tables

The BCTs for the S-boxes σ_0 and σ_1 are given below.

a\b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	6	0	0	0	0	2	0	4	6	2	2	0	0	0	2
2	16	4	0	0	0	2	0	2	6	6	0	0	0	2	2	0
3	16	6	2	2	2	0	0	0	6	4	0	0	2	0	0	0
4	16	0	4	4	0	0	0	0	0	2	0	2	10	0	6	4
5	16	0	4	4	10	0	4	6	0	2	2	0	0	0	0	0
6	16	2	0	2	0	2	0	2	0	0	2	2	0	0	2	2
7	16	2	2	0	0	0	2	2	0	0	2	2	0	2	2	0
8	16	0	10	0	4	0	6	0	0	2	0	0	4	2	4	0
9	16	0	0	0	0	2	2	0	2	0	0	2	2	2	2	2
10	16	0	0	10	4	2	0	4	0	2	0	0	4	0	0	6
11	16	0	0	0	2	2	2	2	2	0	2	0	0	2	0	2
12	16	0	6	0	0	0	2	0	2	0	2	2	4	2	4	0
13	16	2	4	0	6	2	4	2	0	0	2	0	0	2	0	0
14	16	2	0	4	0	2	0	0	0	0	0	2	6	2	2	4
15	16	0	0	6	4	2	0	4	2	0	2	2	0	0	0	2

Table 3.6: BCT for σ_0

a\b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	0	6	0	4	0	6	0	0	2	0	0	2	0	2	2
2	16	0	4	2	6	0	6	2	0	0	0	2	0	2	0	0
3	16	2	6	0	6	2	4	0	2	0	2	0	0	0	0	0
4	16	0	0	0	0	10	2	0	4	0	4	0	0	6	2	4
5	16	10	0	0	0	0	2	0	4	4	4	6	2	0	0	0
6	16	0	2	2	0	0	0	0	0	0	2	2	2	2	2	2
7	16	0	2	0	0	0	0	2	2	2	0	2	2	2	2	0
8	16	4	0	0	0	4	2	2	10	6	0	0	0	4	0	0
9	16	0	0	2	2	2	0	2	0	2	0	0	0	2	2	2
10	16	4	0	2	0	4	2	0	0	0	10	4	0	0	0	6
11	16	2	0	2	2	0	0	2	0	2	0	2	2	0	0	2
12	16	0	0	0	2	4	0	2	6	2	0	0	2	4	2	0
13	16	6	2	2	0	0	0	2	4	4	0	2	2	0	0	0
14	16	0	2	2	0	6	0	2	0	0	4	0	0	2	2	4
15	16	4	0	2	2	0	0	0	0	0	6	4	2	0	2	2

Table 3.7: BCT for σ_1

3.7.4 Incompatibility Analysis

Using an automated tool to check for incompatibilities in the Saturnin S-box, we found 822 cases where the boomerang switch is impossible despite the existence of valid differential trails in the upper and lower parts. These are cases where $DDT(\Delta, \beta) > 0$ and $BCT(\Delta, \nabla) > 0$, but $BDT(\Delta, \beta, \nabla) = 0$. The incompatibility analysis is done on the full cipher not just the toy.

Some examples of such incompatibilities are:

- $\Delta = 1, \beta = 1, \nabla = 2$: $DDT[1][1] = 2, BCT[1][2] = 5$, but $BDT[1][1][2] = 0$.
- $\Delta = 1, \beta = 1, \nabla = 5$: $DDT[1][1] = 2, BCT[1][5] = 5$, but $BDT[1][1][5] = 0$.
- $\Delta = 1, \beta = 1, \nabla = 8$: $DDT[1][1] = 2, BCT[1][8] = 4$, but $BDT[1][1][8] = 0$.

These findings highlight the importance of checking the BDT to ensure the validity of boomerang trails.

CHAPTER 4

Hardware Implementation

4.1 One Round Implementation in Verilog

This section presents a flattened hardware implementation of a toy version of the Saturnin cipher round. The design is fully combinational, combining the nonlinear S-box layer, MDS diffusion, and bit permutation layers into a single flattened Verilog module. This approach eliminates hierarchical overhead, facilitating easier synthesis and accurate area benchmarking.

4.1.1 Design Description

The module `ToySaturninRound_Packed_Flat` takes a 32-bit input state (8 nibbles) and produces a 32-bit output state after one full cipher round. The round function follows the structure:

$$\text{S-box Layer} \rightarrow \text{MDS} \rightarrow \text{S-box Layer} \rightarrow \text{SR\_sheet} \rightarrow \text{MDS} \rightarrow \text{INV\_SR\_sheet}$$

The flattened design incorporates the following components:

- **APN_ANF (S-box):** A 4-bit Almost Perfect Nonlinear S-box described in Algebraic Normal Form (ANF) for nonlinearity.
- **Saturnin_MDS:** An 8×8 binary MDS matrix operating on nibbles for diffusion.
- **SR_sheet & INV_SR_sheet:** Bit permutation layers rearranging bits of nibbles 4–7.

4.1.2 Synthesis Environment

The design was synthesized using the Cadence Genus™ Synthesis Solution (Version 20.11-s111_1) with the `uk65lscllmvbbh_120c25_tc` technology library. The operating condition was set to `balanced_tree` with `top` wireload mode.

S. No.	Cipher Name / S-box	LUT	ANF	DBN	Bit-Width	Alg. Degree	NAND2 GE
1	Keccak	20.5 GE	20.5 GE	2	5	2	1.44
2	Gimli	8.75 GE	8.75 GE	2	3	2	1.44
3	Qarmav2	26 GE	26 GE	2	4	3	1.44
4	Elephant	23.25 GE	24 GE	2	4	3	1.44
5	Romulus	20.75 GE	20.75 GE	2	4	3	1.44
6	Twinkle	22.25 GE	22.75 GE	2	4	3	1.44
7	Saturnin	24.75 GE	25 GE	2	4	3	1.44

Table 4.1: Comparison of S-box Hardware Complexity

CHAPTER 5

Automated Tools

5.1 1 Round MILP Model

This section describes the Mixed Integer Linear Programming (MILP) model used to analyze the differential properties of the Toy Saturnin cipher. The objective is to find the minimal-weight differential trail by encoding the cipher's operations as linear constraints.

5.1.1 Overview

The model represents the cipher structure:

$$\text{S-box} \rightarrow \text{MDS} \rightarrow \text{S-box} \rightarrow \text{SR\_slice} \rightarrow \text{MDS} \rightarrow \text{inv\_SR\_slice}$$

Each round operates on a 32-bit state (8 nibbles). The S-box is the only non-linear component, while other layers are linear permutations or XOR operations.

5.1.2 Phase 1: DDT-based S-box Encoding

In the initial approach, each 4-bit S-box was modeled using its Differential Distribution Table (DDT).

- **Idea:** For each possible input difference Δx and output difference Δy, if the pair $(\Delta x, \Delta y)$ is invalid (probability 0), a linear constraint is added to forbid it.
- **Implementation:**

1. Enumerate all 16×16 difference pairs.
2. Identify invalid transitions from the DDT.
3. For each invalid pair, add a constraint: $\sum \text{bit\_mismatches}(x, y) \leq 7$.

- **Result:** While correct, this method is inefficient, generating hundreds of constraints per S-box (approx. 1000 constraints per layer), leading to slow solve times.

5.1.3 Phase 2: Convex Hull S-box Representation

To improve efficiency, the DDT constraints were replaced with a convex-hull polyhedral encoding.

- **Concept:** Each valid $(\Delta x, \Delta y)$ pair is treated as a point in an 8-dimensional space. The convex hull of all valid points is computed:

$$P = \text{conv}\{(\Delta x, \Delta y) \mid \text{valid pairs}\}$$

  This polytope is expressed as a set of linear inequalities $A \cdot v \leq b$, where $v = [\Delta x, \Delta y]$.

- **Generation:** SageMath's polyhedral geometry tools were used to compute the facets of this polytope.

- **Advantages:** This method significantly reduces the number of constraints to approximately 40-50 per S-box, resulting in a more compact model and faster solve times while maintaining exactness.

5.1.4 Summary of Approaches

Method	**Constraints per S-box**	**Performance**
DDT-based	$\sim 150 - 200$	Slow, large model
Convex Hull	$\sim 40 - 50$	Fast, compact model

Table 5.1: Comparison of MILP Encoding Methods

5.1.5 MILP Results

The MILP model was used to analyze the differential properties of the S-box and the full round function.

S-box Analysis

For the S-box alone, the model found:

- **Optimal Branch Number:** 2.0
- **Differential:** $\Delta x = 0010(2) \rightarrow \Delta y = 0001(1)$
- **Objective:** Best objective 2.0, gap 0.00%.

Full Differential Trail

For the full round function (S-box → MDS → S-box → SR_slice → MDS → inv_SR_slice), the model found an optimal trail with weight 4.0.

Trail Details:

- **Objective:** 4.0 (wt(input) + wt(output))
- **Time to find:** approx. 45 seconds

State Evolution (Hexadecimal):

```
Layer 0 (Input):      [0, 0, 0, 4, 0, 0, 4, 8]
Layer 1 (S-box):      [0, 0, 0, 2, 0, 0, F, 2]
Layer 2 (MDS):        [F, 0, 2, 0, F, 0, D, F]
Layer 3 (S-box):      [8, 0, 8, 0, 4, 0, 4, 4]
Layer 4 (SR_slice):   [8, 0, 8, 0, 8, 0, 8, 8]
Layer 5 (MDS):        [0, 0, 0, 0, 0, 0, 8, 0]
Layer 6 (Output):     [0, 0, 0, 0, 0, 0, 4, 0]
```

5.2 1 Round SAT Solver

This section describes a MiniZinc model for analyzing differential trails in a toy version of the Saturnin block cipher. The model represents each round of Saturnin as a Boolean constraint system and minimizes the number of active S-boxes to identify the weakest possible differential trail.

5.2.1 Overview

This project builds a toy 32-bit Saturnin cipher, reduced to:

- 8 nibbles ($x_0 \dots x_7$)
- 4 bits per nibble

- One complete encryption structure:

```
S-box -> MDS -> S-box -> SR (ShiftRows variant) -> MDS -> inv_SR
```

Two variants exist:

- `SR_slice`: performs permutation $abcd \rightarrow badc$
- `SR_sheet`: performs permutation $abcd \rightarrow cdab$

Each variant affects diffusion differently, allowing direct comparison of their avalanche behavior.

5.2.2 Objective

The objective is to find the **minimum number of active S-boxes** that can occur in one round, and trace **how a single active nibble diffuses** through linear and non-linear layers. This is a standard step in differential and linear cryptanalysis.

5.2.3 Files

File	Description
`saturnin_slice.mzn`	Uses `SR_slice` permutation ($abcd \rightarrow badc$)
`saturnin_sheet.mzn`	Uses `SR_sheet` permutation ($abcd \rightarrow cdab$)

Table 5.2: MiniZinc Model Files

5.2.4 How to Run

Install MiniZinc and run either file using the command line or IDE:

```
minizinc saturnin_sheet.mzn
```

Example output:

```
=== Toy Saturnin (SR_sheet version) ===

Input state (x0..x7, 4 bits each):
[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

```
State after 1st S-box layer:
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

State after 1st MDS layer:
[0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]

...

Active S-boxes after 1st S-box layer: 1
Active S-boxes after 2nd S-box layer: 4
Active S-boxes after final step: 5
Final active flags: [1,1,1,0,1,0,0,1]
```

5.2.5 Cipher Structure

S-box Layer

The 4-bit non-linear substitution box:

```
S(x) = [0, 6, 14, 1, 15, 4, 7, 13, 9, 8, 12, 5, 2, 10, 3, 11]
```

Each active nibble (non-zero input) counts as one **active S-box**.

MDS Layer

A linear diffusion step modeled by an 8×8 binary matrix over GF(2):

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Each bit of each nibble is transformed linearly according to this matrix, spreading active bits across nibbles.

SR Variants

Both apply **only on** $x_4 \dots x_7$, the upper half of the state. The inverse functions restore the original bit order after the second MDS.

Variant	Mapping	Description
SR_slice	$abcd \to badc$	Swaps adjacent bits
SR_sheet	$abcd \to cdab$	Rotates nibble left by two bits

Table 5.3: ShiftRows Variants

5.2.6 Example Trail (Decimal Nibble View)

Below is a **representative minimal trail** from the `SR_sheet` version, converted from bits → nibbles → decimal for easier interpretation:

Step	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
Input	0	0	0	1	1	0	0	0
After 1st S-box	0	0	0	0	1	0	0	0
After 1st MDS	0	1	0	2	0	1	0	0
After 2nd S-box	0	2	0	7	0	3	0	0
After SR_sheet	0	2	0	7	12	9	0	0
After 2nd MDS	3	5	12	6	7	8	9	10
After inv_SR_sheet	3	5	12	6	9	7	10	8

Table 5.4: Example Differential Trail

- **Final number of active S-boxes:** 5
- **Initial active S-boxes:** 1
- **Diffusion ratio:** $\times 5$ increase after one round

5.2.7 Interpreting the Output

Field	Meaning
`Input state`	The Boolean (or decimal) representation of $x_0 \dots x_7$
`State after X`	Bit patterns showing diffusion per step
`Active S-boxes`	Count of non-zero nibble inputs at each S-box layer
`Final active flags`	Binary vector marking which S-boxes were active

Table 5.5: Output Interpretation

5.2.8 Extending the Model

You can:

- Chain multiple rounds by reusing `after_inv_sr` as the next `state_in`.
- Modify the S-box table or MDS matrix to simulate new ciphers.
- Change the objective from `minimize` to `maximize` to explore *maximal* diffusion.

CHAPTER 6

Discussion

6.1 Why is it Lightweight?

Saturnin is designed to be a post-quantum block cipher, meaning it aims to provide security against attackers equipped with quantum computers. Typically, post-quantum security requires larger key sizes (e.g., 256 bits) to resist Grover's algorithm, which provides a quadratic speedup for key search.

Despite this heavy requirement, Saturnin achieves "lightweight" characteristics through several design choices:

- **Efficient S-box:** It uses 4-bit S-boxes, which are very compact in hardware (approx. 25 GE) compared to larger 8-bit S-boxes like AES.
- **Bitslice Implementation:** The cipher is designed to be implemented in a bitsliced manner, allowing for high parallelism and efficiency on software platforms.
- **Simple Linear Layer:** The MDS matrix and ShiftRows operations are chosen to be computationally inexpensive while providing sufficient diffusion.

Thus, "lightweight" in this context refers to the implementation cost and efficiency relative to the high security level (256-bit security) it provides. It bridges the gap between lightweight cryptography (usually 80-128 bit security) and high-security requirements.

6.2 Philosophy of Saturnin

The design philosophy of Saturnin centers on the "Super-round" structure, which combines two rounds to achieve rapid diffusion and confusion.

6.2.1 Role of SR, MDS, and Inverse SR

The combination of ShiftRows (SR), MDS, and Inverse ShiftRows (inv_SR) in the super-round is critical for diffusion:

- **S-box Layer:** Provides non-linearity and local confusion within nibbles.
- **MDS Layer:** Mixes columns, ensuring that a change in one nibble affects all nibbles in the same column.
- **SR_sheet / SR_slice:** These permutations move nibbles to different columns.
- **Inverse Operations:** The use of inverse operations in the second part of the super-round (e.g., inv_SR) interacts with the forward operations to maximize the spread of active nibbles.

Specifically, the sequence $MDS \rightarrow SR \rightarrow MDS \rightarrow inv\_SR$ ensures that differences are not just mixed within columns but are spread across the entire state ("sheets" and "slices") very quickly. Our MILP analysis confirmed that this structure leads to a high number of active S-boxes (branch number of the super-round is high), providing strong resistance against differential attacks.

CHAPTER 7

Conclusion

In this thesis, we presented a comprehensive analysis of the Saturnin block cipher, a candidate for post-quantum lightweight cryptography. By developing a scaled-down "Toy Saturnin" variant, we were able to experimentally verify theoretical properties and demonstrate the feasibility of various cryptanalytic attacks.

Our cryptanalytic investigation covered a broad spectrum of techniques. We confirmed the cubic algebraic degree of the S-box and verified the zero-sum property for up to 2 rounds, showing that the cipher behaves non-randomly in the early rounds but quickly achieves high complexity. Differential-Linear analysis revealed strong biases for 1-round characteristics, while Boomerang analysis highlighted the importance of checking for incompatibilities in the S-box, identifying hundreds of invalid switching cases.

The hardware implementation study demonstrated that Saturnin's S-box is highly efficient, requiring approximately 25 Gate Equivalents (GE), which is comparable to other state-of-the-art lightweight S-boxes like those in GIFT or SKINNY, yet it offers a higher algebraic degree of 3. This confirms Saturnin's design goal of balancing post-quantum security requirements with lightweight implementation constraints.

Furthermore, the application of automated tools such as MILP and SAT solvers proved invaluable. The MILP model successfully found the optimal branch number of 2.0 for the S-box and constructed a full differential trail with a weight of 4.0, providing concrete bounds on the cipher's differential resistance. The convex hull technique significantly optimized the MILP solving process.

In conclusion, Saturnin stands as a robust and efficient design. Its "super-round" structure, combining MDS and ShiftRows operations, ensures rapid diffusion, while its S-box provides a good trade-off between nonlinearity and hardware cost. Our study validates its suitability for lightweight applications requiring long-term security against quantum threats. Future work could extend this analysis to more rounds and explore the resistance against other advanced attacks like integral or invariant subspace attacks on the full version of the cipher.

About the Author

I'm Kriti, and I got interested in cryptography during Dhiman sir's OS course, which also got me into low-level systems in general. I took the crypto course afterward and really enjoyed it, so I chose Lightweight Cryptography this semester. I didn't expect to learn things like Verilog, Cadence, or hardware-focused design, but I ended up liking that side of crypto too.

I've also taken ML courses and enjoy that area as well. Overall, I'm curious about a lot of things and like exploring different fields. If you want to discuss anything related, feel free to reach out to me at the email below.

- kritia@iitbhilai.ac.in

Appendix

Convex Hull Inequalities for S-box

The following 222 inequalities describe the convex hull of the valid differential transitions for the Saturnin S-box. These are used in the MILP model.

```
[0, -1, 0, 0, 0, 0, 0, 0] <=  -1
[-1, 0, 0, 0, 0, 0, 0, 0] <=  -1
[0, 0, -1, 0, 0, 0, 0, 0] <=  -1
[0, 0, 0, -1, 0, 0, 0, 0] <=  -1
[0, 0, 0, 0, -1, 0, 0, 0] <=  -1
[0, 0, 0, 0, 0, -1, 0, 0] <=  -1
[0, 0, 0, 0, 0, 0, 0, -1] <=  -1
[-1, 1, -1, 0, -1, -1, 0, -1] <=  -4
[-1, -1, -2, -2, 1, -2, -1, 1] <=  -7
[-1, -1, 0, -1, 1, 0, -1, -1] <=  -4
[-1, 0, -1, -1, 1, -1, -1, 0] <=  -4
[0, 1, -1, -1, 0, -1, -1, -1] <=  -4
[1, 1, -1, -2, -1, -2, -2, -1] <=  -7
[-1, 1, -1, 0, 0, -1, -1, -1] <=  -4
[0, -1, -1, -1, 0, -1, -1, 1] <=  -4
[1, 0, -1, -1, -1, -1, -1, 0] <=  -4
[-1, 1, -2, -1, 1, -1, -2, -1] <=  -6
[-2, 1, -2, 1, -1, -1, -1, -2] <=  -7
[0, 0, 0, 0, 0, 0, -1, 0] <=  -1
[-1, 0, -1, -1, 1, 0, -1, -1] <=  -4
[-2, -1, -1, -1, 1, 1, -2, -2] <=  -7
[-1, 1, -1, 0, 1, 1, 1, 0] <=  -1
[-1, 1, -1, 0, 1, 1, 0, -1] <=  -2
[0, -1, 1, 0, -1, -1, 1, -1] <=  -3
[-3, 1, -2, 1, 2, 2, -1, -3] <=  -6
[-1, 1, -1, -2, -1, 1, 2, -2] <=  -5
```

```
[-1, -1, 0, -1, 1, 1, 2, 2] <=  -1
[-1, -1, -1, -1, -2, 2, 1, -1] <=  -5
[-1, 0, 0, 1, -1, 1, -1, 1] <=  -2
[0, 0, 0, 0, 1, 0, 0, 0] <=  0
[0, 0, 0, 0, 0, 1, 0, 0] <=  0
[0, 0, 0, 0, 0, 0, 1, 0] <=  0
[-1, -1, 0, -1, 0, 1, 1, 1] <=  -2
[-1, -1, 1, -1, -1, 0, 1, 0] <=  -3
[-2, 2, -2, -1, 1, 2, 1, -1] <=  -4
[0, 0, -1, 0, 1, 1, 1, 1] <=  0
[1, -2, 0, -1, 2, 0, 1, 1] <=  -1
[0, 0, 0, -1, 1, 1, 1, 1] <=  0
[-1, -1, 0, -1, 2, 2, 3, 3] <=  0
[1, -1, -1, 0, 2, 1, 1, 1] <=  0
[1, -1, -1, 1, 0, 1, 0, -1] <=  -2
[-1, 0, 0, 0, 1, 1, 1, 1] <=  0
[-1, -1, 1, 1, 0, -1, -1, 0] <=  -3
[-2, -1, 2, -3, -2, 1, 3, 1] <=  -5
[-1, -1, 1, 1, -1, 0, -1, 0] <=  -3
[0, 0, -1, 1, 1, 1, 1, 0] <=  0
[1, -1, -1, 0, 0, 1, 1, -1] <=  -2
[0, 1, 1, 1, 1, 0, -1, -1] <=  -1
[1, 1, 0, -1, 0, 1, 1, 1] <=  0
[2, 2, 1, 1, 1, 0, 1, -2] <=  0
[1, 1, 1, 1, 0, -1, 0, 0] <=  0
[1, 2, 2, 2, 1, 0, -1, -1] <=  0
[0, -1, 0, -1, 1, 0, 1, 1] <=  -1
[-1, 0, 2, -2, -1, 1, 2, 1] <=  -2
[-1, 0, 1, 1, -1, 0, -1, 1] <=  -2
[3, 1, 3, 1, -1, -2, 1, -1] <=  -1
[1, -1, 0, 1, 1, 1, -1, 0] <=  -1
[-2, 0, 1, 2, -2, 1, -2, 1] <=  -4
[-2, -1, 1, -2, -2, 1, 2, 0] <=  -5
[2, -1, 3, 1, 1, -3, 3, -1] <=  -2
[1, 0, 1, 1, -1, -1, 0, -1] <=  -2
[-1, 1, 3, -3, -1, 1, 3, 2] <=  -2
[1, 3, 1, 2, -1, 1, -2, 1] <=  0
[2, 1, 2, 1, -1, -1, 0, -1] <=  -1
[3, 2, 2, 1, 1, -1, 2, -2] <=  0
[3, 2, 3, 2, -1, -1, 0, -1] <=  0
[1, 0, 1, 0, 0, -1, 1, -1] <=  -1
```

```
[1, -1, 1, 1, 2, -1, -1, 2] <=  -1
[1, 1, 2, 1, 1, 1, -1, -1] <=  0
[1, 2, 1, 2, -1, 1, -1, 0] <=  0
[2, 1, 2, 1, -1, -1, 1, 0] <=  0
[1, 0, 1, -2, 1, 2, 2, 1] <=  0
[0, -1, -2, 2, 1, 2, 1, -1] <=  -2
[2, 1, 2, 1, 0, -1, 1, -1] <=  0
[1, 1, -1, 0, 0, 1, 0, 1] <=  0
[-1, -1, 1, 1, 0, -1, 0, -1] <=  -3
[1, 2, 3, 2, 1, 1, -2, -1] <=  0
[1, 1, 1, 0, 1, 0, 1, -1] <=  0
[-1, 1, -1, 0, 2, 2, 2, 1] <=  0
[1, -2, -3, 2, 1, 3, 1, -2] <=  -4
[1, 0, 1, 1, 1, 1, -1, 0] <=  0
[1, 1, 1, 1, 0, 0, -1, 0] <=  0
[-1, 2, 1, 3, -1, 3, -3, -1] <=  -3
[-1, 1, 1, 0, 1, 1, 0, 1] <=  0
[0, -1, -1, 1, 1, 1, 0, 0] <=  -1
[1, 2, 1, 1, -1, 0, -1, 1] <=  0
[0, -1, 0, 0, 1, 1, 1, 1] <=  0
[-1, 1, 1, 2, -1, 2, -2, 0] <=  -2
[1, 2, 1, 0, -2, -1, 0, 1] <=  -1
[2, 1, 2, -1, 2, 1, 1, -1] <=  0
[2, 2, -1, -1, -1, 1, 1, 1] <=  -1
[1, 1, 0, 1, 0, 0, -1, 1] <=  0
[1, 1, 1, 0, 1, 1, 0, -1] <=  0
[1, 1, 0, 0, -1, 0, 0, 1] <=  0
[-1, 1, -1, 2, -2, 2, -1, 1] <=  -3
[2, 0, 2, 1, -1, -2, 1, -1] <=  -2
[-1, 0, 1, 1, -1, 1, -1, 0] <=  -2
[1, 0, 1, 1, 0, -1, 1, 0] <=  0
[1, 1, 1, 1, 0, 0, 0, -1] <=  0
[0, 0, 1, 0, 0, 0, 0, 0] <=  0
[-1, 0, 1, -1, -1, 1, 1, 0] <=  -2
[1, 0, -1, 1, -1, 1, -1, -1] <=  -3
[0, -1, 1, 1, -1, -1, 0, -1] <=  -3
[1, 3, 1, -1, -3, -1, 1, 2] <=  -2
[2, 2, 2, 1, -2, -1, 1, 1] <=  0
[1, 2, 1, 1, -1, -1, 0, 1] <=  0
[4, 2, 4, 2, -1, -2, 1, -1] <=  0
[2, 1, 1, 1, 1, -1, 1, -1] <=  0
```

```
[1, 0, 0, 0, 0, 0, 0, 0] <=  0
[2, 2, 2, 1, -1, -1, 0, 0] <=  0
[0, 0, 0, 1, 0, 0, 0, 0] <=  0
[-1, -1, 1, -2, -1, 1, 2, 1] <=  -3
[1, 1, 1, 1, -1, -1, 1, 1] <=  0
[1, 1, 0, -1, -1, -1, -1, 0] <=  -3
[-2, 1, 1, 3, -2, 2, -3, 1] <=  -4
[-1, 1, -1, -1, -1, -1, 1, 0] <=  -4
[0, -1, 0, -1, 2, 1, 2, 2] <=  0
[1, 2, 2, 2, -1, -2, 1, 2] <=  0
[-2, 1, 2, -1, 1, -1, 2, 1] <=  -2
[-1, 1, -1, 0, 1, 0, -1, -1] <=  -3
[-1, -1, 0, -1, -1, 1, 1, 0] <=  -3
[0, 1, 0, -1, -1, -1, 1, 1] <=  -2
[1, 2, 0, 1, -1, 1, -1, 1] <=  0
[2, 1, 3, -1, 2, -1, 3, -3] <=  -2
[-1, -1, -1, 1, -1, -1, 1, 1] <=  -4
[0, 1, 0, 0, 0, 0, 0, 0] <=  0
[1, -2, -1, 2, 1, 2, -1, -1] <=  -3
[-1, 1, 0, 1, 0, 1, -1, -1] <=  -2
[-1, -1, -3, 3, 2, 3, 1, -1] <=  -3
[0, 1, -1, 1, 0, 1, 1, 1] <=  0
[1, 2, -1, 1, -1, 1, 0, 2] <=  0
[2, 3, -1, 1, -1, 1, -1, 2] <=  0
[3, 3, -1, -1, -1, 2, 1, 2] <=  0
[1, 1, 0, 1, 1, 0, 1, -1] <=  0
[1, 1, 1, -3, 1, 3, 3, 2] <=  0
[-1, 1, 0, 1, 1, 0, -1, -1] <=  -2
[-1, -1, 1, -2, 0, 1, 2, 2] <=  -2
[0, 0, 0, 0, 0, 0, 0, 1] <=  0
[0, 0, -1, -1, -1, 1, 1, -1] <=  -3
[1, 1, 2, 2, 0, -2, 1, 1] <=  0
[0, -1, -1, -1, 1, -1, 0, 1] <=  -3
[-1, 1, -1, 1, 0, 0, -1, -1] <=  -3
[-1, 0, -1, 0, 1, 1, -1, -1] <=  -3
[-3, 2, 3, -1, 1, -1, 3, 2] <=  -2
[1, -1, 0, -1, 2, 1, 1, 1] <=  0
[-1, 1, 1, 0, 0, -1, 1, 1] <=  -1
[2, -1, 0, 1, 2, 1, -1, 1] <=  0
[1, 0, 0, 1, 1, 0, -1, 1] <=  0
[1, -2, 0, -1, 3, 1, 2, 2] <=  0
```

```
[-1, 1, 0, 1, 1, 0, 1, 1] <=  0
[-1, 1, 1, -1, 1, 1, 1, 2] <=  0
[-1, 2, 1, 1, 1, -1, 2, 2] <=  0
[0, 1, 1, 1, 0, -1, 1, 1] <=  0
[1, -1, 2, 1, 0, -2, 2, -1] <=  -2
[1, -2, 3, 1, -1, -3, 2, -2] <=  -5
[1, 0, 1, -1, 1, 0, 1, -1] <=  -1
[1, -1, 0, 0, 1, 0, 0, 1] <=  0
[1, -1, -2, 1, 1, 2, 1, -1] <=  -2
[0, 1, 0, 1, -1, 1, -1, 0] <=  -1
[1, 1, 2, 2, 1, 1, -2, 0] <=  0
[0, 1, 1, 1, 0, 1, -1, 0] <=  0
[-1, 1, 1, 2, 2, -1, -2, -2] <=  -4
[-1, -2, 1, -1, 2, -1, -2, -2] <=  -7
[2, -1, 1, 2, 3, -1, -1, 3] <=  0
[-1, 0, -1, 1, 1, 1, 0, -1] <=  -2
[-1, 0, 1, -1, 1, 1, 2, 2] <=  0
[1, -1, 2, 1, -1, -2, 1, -1] <=  -3
[-1, 1, 1, -1, -1, -1, 1, 2] <=  -2
[-2, 1, 1, 0, 2, 1, 1, 2] <=  0
[-1, 1, -2, 1, 2, 2, 3, 1] <=  0
[0, 1, 1, -1, 1, -1, 1, -1] <=  -2
[-2, 1, -2, 0, 2, 1, -1, -2] <=  -5
[-1, 0, 1, -1, 0, 0, 1, 1] <=  -1
[-1, -2, 1, 2, -1, -1, -1, -1] <=  -5
[-2, 2, -2, -1, -2, -1, 1, -1] <=  -7
[0, -1, 1, -1, 1, -1, -1, -1] <=  -4
[0, 1, -2, 1, 1, 2, 2, 1] <=  0
[-1, 1, 1, 2, -1, 1, -2, 1] <=  -2
[1, 1, 1, 0, -1, -1, 0, 0] <=  -1
[-1, -1, -1, 1, 0, 1, -1, 1] <=  -3
[1, 0, 1, 1, 1, -1, 0, 1] <=  0
[2, -1, 1, -2, 2, 2, 2, 1] <=  0
[2, 3, 2, 1, -2, -1, 0, 1] <=  0
[0, 1, -1, 1, -1, 1, 0, 1] <=  -1
[0, -1, -1, 0, -1, 1, 1, -1] <=  -3
[1, -1, 1, -1, 1, -1, 0, 0] <=  -2
[-1, -1, 1, -2, 1, 1, 3, 3] <=  -1
[1, 0, -1, 0, 1, 1, 0, 1] <=  0
[0, -1, -1, 1, 2, 2, 1, 1] <=  0
[1, 1, 1, 1, -1, 0, 0, 0] <=  0
```

```
[0, 1, 1, 0, -1, -1, 0, 1] <=  -1
[-1, 1, -1, 1, 1, 1, 2, 1] <=  0
[0, -1, 1, 1, 0, -1, 1, -1] <=  -2
[2, -1, 1, -1, 1, -1, 1, -1] <=  -2
[1, -1, 0, -1, 1, 0, 1, 0] <=  -1
[-1, 0, 0, -1, -1, 1, 1, -1] <=  -3
[2, 1, 3, 3, 1, -3, 1, 1] <=  0
[-2, 2, -1, 1, 1, 0, -2, -2] <=  -5
[1, 1, 2, -1, 1, -1, 2, -2] <=  -2
[2, -3, 1, -1, 3, -1, 1, 1] <=  -2
[-1, 1, 1, 0, 1, 0, 1, 1] <=  0
[-1, 1, 1, -1, 1, -1, 1, 0] <=  -2
[2, -1, -1, 1, 3, 1, -1, 2] <=  0
[1, -1, 1, -1, -1, 1, -1, -1] <=  -4
[1, -1, -1, 1, 2, 1, 0, 1] <=  0
[-2, 1, 2, -1, 2, 1, 2, 3] <=  0
[0, -1, 1, -1, 1, 1, 2, 2] <=  0
[0, -2, 2, 1, -1, -2, 1, -2] <=  -5
[-3, 2, -1, 2, 1, 1, -2, -3] <=  -6
[0, 0, 1, -1, 0, 1, 1, 1] <=  0
[-1, -1, 1, -2, 2, 2, 4, 4] <=  0
[1, -2, -2, 1, 0, 2, 1, -2] <=  -4
[1, -1, 1, -1, 1, 1, 1, 1] <=  0
[-2, -1, 2, 1, 1, -2, -1, -1] <=  -5
[1, 0, 1, -1, 1, 1, 1, 0] <=  0
[1, -2, 1, -1, 2, 1, 2, 2] <=  0
[1, -1, -2, 1, 3, 2, 1, 1] <=  0
[0, -1, -1, 1, 0, 1, 1, -1] <=  -2
[-1, -1, 0, 1, -1, -1, -1, -1] <=  -5
[-1, 0, -1, 1, -1, -1, -1, -1] <=  -5
[-1, -1, -1, -1, 0, 1, -1, -1] <=  -5
[-1, -1, -1, -1, -1, 1, 0, -1] <=  -5
```

Brownie Points

Encrypted Chat Server

As an additional project, we developed a secure chat server application to demonstrate the practical usage of lightweight ciphers. The system features a client-server architecture where:

- One client encrypts messages using the **Twinkle** lightweight cipher.
- The other client encrypts messages using the **Saturnin** block cipher.

This setup demonstrates interoperability and the application of different lightweight cryptographic primitives in a real-time communication scenario.

Bibliography

[BODKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. Differential-linear connectivity table. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–31. Springer, 2019.

[CDL+20] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. Saturnin: a suite of lightweight symmetric algorithms for post-quantum cryptography. *IACR Transactions on Symmetric Cryptology*, pages 371–411, 2020.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 683–714. Springer, 2018.

[LH94] Susan K Langford and Martin E Hellman. Differential-linear cryptanalysis. In *Annual International Cryptology Conference*, pages 17–25. Springer, 1994.

[Wag99] David Wagner. The boomerang attack. In *Fast Software Encryption*, pages 156–170. Springer, 1999.