On the Security of Rotational (Non-)linearity in Sbox

A case study of Skyscraper-v2

Yadi Zhong

Auburn University, Auburn AL 36849, USA
yadi@auburn.edu

Abstract. Recently, zero-knowledge proof protocols have gained much popularity due to the adoption in blockchain applications, e.g., zero-knowledge virtual machines. However, using the current standardized hash functions inside the generation of zero-knowledge proofs would incur much overhead in proof size, as well as prover and verifier's runtime. In the past few years, various circuit-friendly hash functions has been proposed. Skyscraper-v2 is one example of such hash functions applying the split-and-lookup approach for better performance. In this paper, we expand the linear approximation definitions by extending it with circular shifts embedded in the approximation. Specifically, we consider the rotation of bits at both the input and output sides. We demonstrate it with Skyscraper-v2 Sbox. It allows us to better capture the recurring sequence in nonlinear Skyscraper-v2 SBox.

Keywords: Arithmetization-Oriented hash functions · Sbox · linear approximation with rotations.

1 Introduction

1.1 Arithmetization-Oriented Hash Functions

Over the past decade, applications involving zero-knowledge proof protocols have gained much popularity due to its adoption in blockchain environment, e.g., zero-knowledge virtual machines. However, using the current standardized hash functions inside the generation of zero-knowledge proofs would incur much overhead in proof size, as well as prover and verifier's runtime. In the past few years, various circuit-friendly hash functions has been proposed. The target is to construct efficiently arithmetized hash function, e.g., under R1CS constraint systems, where these hash functions are also named as arithmetization-oriented designs. MiMCHash [2], Poseidon [19], Neptune [22], Poseidon2 [20], Poseidon2b [16] are based on low-degree permutations. *Rescue-Prime* [28], `Anemoi` [11], *Vision Mark-32* [4] are based on design strategies of either CCZ-equivalence or functions with a high-degree evaluation while low-degree verification. `Reinforced Concrete` [17], Tip5 [29], `Monolith` [18], Skyscraper [12] are focused on introducing lookup tables over a small subset of the underlying large prime field for better performance. They are also called the split-and-lookup constructions.

As arithmetization-oriented hash functions are relevantly new constructions, cryptanalysis efforts are still ongoing. Recently published attacks [1, 3, 6, 7, 15, 21, 23, 24, 27, 30] consider the exploit of algebraic structures of these circuit-friendly designs. One notable attack, the Freelunch attack [8] by Bariant et. al., focuses on solving polynomial systems with Groebner basis on constrained-input constrained-output (CICO) problem. On the other hand, Skyscraper suffered the rebound attack by Antoine Bak [5]. Skyscraper-v2 [13] is later introduced to supersede the original Skyscraper design by increasing 4 additional of square operations both at the beginning and the end of the permutation.

Bak and Perrin [7] first analyzed the security property of split-and-lookup constructions of Skyscraper, `Monolith`, Tip5, `Reinforced Concrete`. They present a method to efficiently compute the Fourier coefficients and the linear approximation probabilities of the Sboxes. Skyscraper, `Monolith`, Tip5, `Reinforced Concrete` all have high probabilities for linear approximations.

For a better construction of the nonlinear component, e.g., Sbox design, in arithmetization-oriented constructions, we ask the the following question:

Can we better capture the linear approximation of Sbox designs?

1.2 Contributions

In this paper, we answer the previous question by investigating the role of circular shift (i.e. bit rotation) in the input and/or output. In particular, we modify the existing definition of linear approximation by allowing the rotation of both input and output. Hence, the paper centers around the following question.

Does the inclusion of circular shift with linear approximation better capture the nonlinearity of Sbox designs?

The contributions of the paper are as follow.

1. We expand the linear approximation definitions by extending it with circular shifts embedded in the approximation. Specifically, we consider the rotation of bits at both the input and output sides.
2. We analyze the Skyscraper-v2 Sbox with absolute weight difference $\mathcal{AWD}$ between input-output pairs and identify the three categories of $\mathcal{AWD}_0^T$, $\mathcal{AWD}_1^T$, and $\mathcal{AWD}_2^T$. Then, we are able to reconcile all three categories with our definition of (generalized) linear approximation with the inclusion of rotational structures.

1.3 Outline

Section 2 covers the preliminaries and gives a brief overview of Skyscraper-v2. Section 3 presents our proposed linear approximation framework on Skyscraper-v2 Sbox. Finally, Section 4 concludes this work.

2 Preliminaries

Notations. In this paper, we denote $\mathbb{F}_q$ as finite field of order q, where q is a prime number or a power of prime. We use $[a]$ the set of integers $\{0, \ldots, a-1\}$ for $a \in \mathbb{N}$. For any vector $v \in \mathbb{F}_q$, the jth element (or bit if $q = 2^k$) is denoted as v_j.

Differential cryptanalysis was first explored by Biham and Shamir [9, 10] in the early 1980s. The core is to consider the distribution (aka the probability) of input-output-differences pairs from a single encryption round, then generalized the observation to multiple rounds by assuming the independence between rounds.

Definition 1 (Difference Distribution Table (DDT)). *Let function $F : \mathbb{F}_q^m \to \mathbb{F}_q^m$, and $\mathbb{F}_q^m$ elements $\Delta_{in}, \Delta_{out}$. The Difference Distribution Table (DDT) contains the number of solutions of the equation*

$$F(x + \Delta_{in}) - F(x) = \Delta_{out}$$

for each given pair of difference $(\Delta_{in}, \Delta_{out})$.

$$DDT_F[\Delta_{in}, \Delta_{out}] = \#\{x \in \mathbb{F}_q^m, F(x + \Delta_{in}) - F(x) = \Delta_{out}\}$$

Definition 2 (Differential Uniformity). *The maximum value in the DDT, δ_F, is the differential uniformity*

$$\delta_F = \max_{\Delta_{in} \neq 0, \Delta_{out}} \#\{F(x + \Delta_{in}) - F(x) = \Delta_{out}\}.$$

Linear cryptanalysis, first introduced by Mitsuru Matsui [25, 26] explored linear relations between the plaintexts and the corresponding ciphertexts, which can be sumarized in Linear Approximation Table (LAT).

Definition 3 (Linear Approximation Table (LAT)). *Let $F : \mathbb{F}_q^m \to \mathbb{F}_q^m$ be a function, and $a, b \in \mathbb{F}_q$. The Linear Approximation Table (LAT) is defined from*

$$LAT_F[a, b] = \begin{cases} \sum_{x \in \mathbb{F}_2^m} (-1)^{a \cdot x + b \cdot F(x)}, & \text{if} \quad q = 2 \\ \sum_{x \in \mathbb{F}_q^m} \exp\left(\frac{2\pi i(\langle a, x\rangle - \langle b, F(x)\rangle)}{q}\right), & \text{if} \quad q > 2. \end{cases}$$

Coefficients in the LAT are also known as the Walsh coefficients.

Definition 4 (Linearity). *Linearity is defined as the module of the highest coefficient, $\mathcal{L}_F$,*

$$\mathcal{L}_F = \max_{a,b \neq 0} |LAT_F[a, b]|.$$

2.1 Description of Skyscraper-v2

We briefly describe the construction of Skyscraper-v2 [13] with its round function and Sbox design. Skyscraper-v2 is based on prime fields $\mathbb{F}_p$, where $p \geq 2^{248}$ and evaluated with the extension field $\mathbb{F}_{p^n} \equiv \mathbb{Z}_p[X]/G(x)$ with irreducible polynomial $G(x)$ of $X^2 + 5$ for $n = 2$ and $X^3 + 2$ (or $X^3 + 3$) for $n = 3$.

Skyscraper-v2 permutation is constructed with 18 round operations,

$$(v_L, v_R) \mapsto S_{17} \circ S_{16} \circ S_{15} \circ S_{14} \circ S_{13} \circ S_{12} \circ B_{11} \circ B_{10} \circ S_9 \circ S_8 \circ B_7 \circ B_6 \circ S_5 \circ S_4 \circ S_3 \circ S_2 \circ S_1 \circ S_0(v_L, v_R)$$

where S_i and B_j are ith round square operation and jth round Bars operation, respectively. Both follow the Feistel round structure.

Square Operation. The operation $S_i : \mathbb{F}_{p^n}^2 \to \mathbb{F}_{p^n}^2$ is a round function operating on the left input v_L with squaring operation and the addition of a pseudo-random constant γ_i

$$(v_L, v_R) \mapsto \left(v_R + \frac{(v_L)^2}{\psi} + \gamma_i, v_L \right).$$

where ψ is a Montgomery constant, where $(2^{256} \bmod p)$ for $p < 2^{256}$ and $(2^{512} \bmod p)$ for other p.

Bars Operation. The Bars operation $B_i : \mathbb{F}_{p^n}^2 \to \mathbb{F}_{p^n}^2$ is defined as

$$(v_L, v_R) \mapsto (v_R + B'(v_L) + \gamma_i, v_L),$$

where γ_i is pseudo-random round constant and the function $B' : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ is built from the following subroutines.

1. *Decomp.* Consider the input $v \in \mathbb{F}_{p^n}$ as $(v_0, v_1, \ldots, v_{n-1}) \in \mathbb{F}_p^n$, then split each $v_j \in \mathbb{F}_p$ into $m = \lceil \log_2(p)/8 \rceil$ 8-bit chunks, i.e.,

   $$v \mapsto (w_0, w_1, \ldots, w_{n \cdot m - 1}) \in (\mathbb{F}_2^8)^m,$$

   where $x_j = \sum_{\ell=0}^{m-1} 2^{8(m-1-\ell)} \cdot w_{j \cdot m + \ell}$ for all $0 \leq j < n$.
2. *Rot*$_{m/2}$. Apply a cyclic rotation to w by $m/2$ elements such that

   $$w'_k \leftarrow w_{(k+m/2) \bmod n \cdot m}, \quad 0 \leq k < n \cdot m.$$

3. *Sbox.* Apply an 8-bit operation T to each w'_k separately:

   $$z_k \leftarrow T(w'_k), \quad 0 \leq k < n \cdot m.$$

   where $T : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ is defined as

   $$T(x) = (x \oplus ((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3))) \lll 1$$

   where the bitwise XOR operation is denoted as $\oplus$, AND operation as *odot*, and circular left shift with $\lll$. Note that T is identical to the Sbox used in `Monolith -64` and `Monolith -31`[1] [14, 18].

[1] `Monolith -31` also has a 7-bit Sbox, $x \mapsto (x \oplus ((\overline{x} \lll 1) \odot (x \lll 2))) \lll 1$.

4. *Comp.* Combine $(z_0, z_1, ..., z_{m \cdot n})$ to $(y_0, ..., y_{n-1})$ with vector $y \in \mathbb{F}_{p^n}$,

$$y_j = \left(\sum_{\ell=0}^{m-1} 2^{8(m-1-\ell)} \cdot z_{j \cdot m + \ell} \right) \bmod p$$

2.2 Original Skyscraper-v2 [13] SBox Analysis

Authors in Skyscraper-v2 [13] presented their analysis on the linear approximation of **Bars** with the following remarks on Sbox T. (see remarks after Lemma 7 of [13].)

Remark 1. The simplified map $T' : x \mapsto x \oplus ((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3))$ over $\mathbb{F}_2^8$ has exactly 48 fixed points, where 34 of them has 0 as the most significant bit.

Remark 2. Next, the circular shift operation $\cdot \lll 1$ is applied on $x \mapsto (x \oplus ((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3))) \lll 1$ over $\mathbb{F}_2^8$ corresponds to the double operation over $\mathbb{Z}_2^8$ when the most significant bit is 0. Then, we can approximate $T : x \mapsto (x \oplus ((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3))) \lll 1$ with $x \mapsto 2 \cdot x$ over $\mathbb{Z}_2^8$ when x is a fixed point of $(\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3))$. The probability that this $x \mapsto 2 \cdot x$ approximation is correct is at least

$$\frac{34}{256} \approx 2^{-2.913}$$

3 Linear Approximation with Rotation

Linear cryptanalysis [26], first applied to DES, is one of the fundamental attacks in symmetric cryptography. It considers the linear approximation and correlations between input and output bits in the known-plaintext setting. In this section, we revisit the linear approximation from Bak and Perrin [7] and extend it with circular shifts embedded in the approximation.

3.1 Analysis of Sbox in Skyscraper-v2

Skyscraper-v2 Sbox design T maps input $x \in \mathbb{F}_2^8$ to $y \in \mathbb{F}_2^8$ with

$$y = T(x) = (x \oplus ((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3))) \lll 1.$$

It is straightforward to show that there are 48 input combinations produce the 0 evaluation for $(\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3)$. This means that $\frac{48}{256} = 3/16$ of input combinations of x maps to output $y = x \lll 1$ under T. Readers can verify it with 256 input-output pairs of Sbox of Tables 2–4, in Appendix A.

Futhermore, suppose we consider the weight of input x, denoted as $wt(x)$, versus the weight of output $y = T(x)$, $wt(y)$ after being evaluated with Sbox T. Define the absolute weight difference of Sbox input-output pair as $aw(x, y) =$

$|wt(x) - wt(y)|$. The distribution of absolute weight difference $aw(x, y)$ is computed as

$$\mathcal{AWD}_i^T = \#\{(x, y) \in \mathbb{F}_2^8 \times \mathbb{F}_2^8 | y = T(x), i = aw(x, y)\}.$$

For Skyscraper-v2 Sbox, the $\mathcal{AWD}_i^T$ is distributed as follows.

$\mathcal{AWD}_0^T$	$\mathcal{AWD}_1^T$	$\mathcal{AWD}_2^T$
72	160	24

For input and output of Sbox T to have the same weight, there are 72 out of 256 with $\mathcal{AWD}_0^T = 72$. Among these $\mathcal{AWD}_0^T = 72$, we have

- 2 fixed points, 00000000 and 11111111, with $y = x$;
- 46 input-output pairs with $y = x \lll 1$ (excluding 00000000 and 11111111);
- 8 input output pairs with $y = x \lll 5$; and
- the rest 16 input-output pairs cannot be categorized with circular shift operation between x and y.

The majority entries in Sbox T have the absolute weight different of 1, where $\mathcal{AWD}_1^T = 160$. From Table 4 in Appendix A, all input-output pairs from $\mathcal{AWD}_1^T$ can be considered as the following form,

$$y = (x \oplus oh) \lll 1,$$

where oh is a one-hot vector in $\mathbb{F}_2^8$ with weight $wt(oh) = 1$, i.e., oh has a single bit 1 and seven 0s.

Similarly, for the rest of input-output pairs with $\mathcal{AWD}_2^T$, input vector x could have a weight difference of ± 2 than output vector y. Specifically, by observing all 24 Sbox entries with $\mathcal{AWD}_2^T$ from Table 4, we can summarize these as

$$y = (x \oplus th) \lll 1,$$

where th is a two-hot vector in $\mathbb{F}_2^8$ with weight $wt(th) = 2$, i.e., th has two 1s and six 0s. Additionally, the corresponding bit locations of x in the non-zero ith, jth bit of vector vh with $v_i = 1, v_j = 1$ need to satisfy $x_i = x_j$.

From observations of above, most of the entries in Sbox T of Skyscraper can be modeled with linear and circular bit-shift operations. However, there are 16 input-output pairs in $\mathcal{AWD}_0^T$ which are not a direct shift of input x. In the subsequent sections, we explore how to better approximate the relations of Sbox input-output with both linear and bit-rotation steps.

3.2 Linear Approximation

In general, the linear approximation of a function f mapping $\mathbb{F}_p^n \to \mathbb{F}_p^m$ and the linear probability are used extensively in linear cryptanalysis [7]. They are defined as follow.

Definition 5 (Linear Approximation). *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$. For $a \in \mathbb{F}_p^m, b \in \mathbb{F}_p^n, c \in \mathbb{F}_p$, the linear approximation is the number of solutions satisfying $a \cdot f(v) = b \cdot v + c$,*

$$L^f(a, b, c) = \#\{v \in \mathbb{F}_p^n | a \cdot f(v) = b \cdot v + c\}.$$

With linear approximation, the linear probability of function f with masks a, b, c is defined over the input space $\mathbb{F}_p^n$ as

$$\Pr_{v \in \mathbb{F}_p^n}[a \cdot f(v) = b \cdot v + c] = L^f(a, b, c)/p^n.$$

For the **Bars** operation in Skyscraper-v2, its linear approximation can be considered as

$$L^B_{a,b,c} = \#\{v \in \mathbb{F}_p | a \cdot B(v) = b \cdot v + c\}.$$

Subsequently, the linear approximation of ith Sbox T_i with input v_i is

$$L^T_{a,b,c_i} = \#\{v_i \in \mathbb{F}_2^8 | a \cdot T(v_i) = b \cdot v_i + c_i\}.$$

It is straightforward that the lower bound of $L^B_{a,b,c}$ can be expressed as

$$L^B_{a,b,c} \geq \prod_{i=1}^{n} L^T_{a,b,c_i}.$$

3.3 Linear Approximation with Rotation and its Generalization

With results from Section 3.1, most of the entries in Sbox T of Skyscraper can be modeled with linear and circular bit-shift operations. Therefore, we extend the notion of linear approximation to include the condition of circular rotating the input.

Definition 6 (Linear Approximation with Rotation). *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$. For $a \in \mathbb{F}_p^m, b \in \mathbb{F}_p^n, c \in \mathbb{F}_p, d \in \mathbb{F}_{n \log(p)}$, the linear approximation with rotation is the number of solutions satisfying $a \cdot f(v) = b \cdot (v \lll d) + c$,*

$$L^f(a, b, c, d) = \#\{v \in \mathbb{F}_p^n | a \cdot f(v) = b \cdot (v \lll d) + c\}.$$

For Sbox $T : \mathbb{F}_2^8 \to \mathbb{F}_2^8$, the linear approximation with rotation is defined as

$$L^T_{a,b,c,d} = \#\{x \in \mathbb{F}_2^8 | a \cdot T(x) = b \cdot (x \lll d) + c\},$$

where $a, b, c \in \mathbb{F}_2^8$ and $d \in \mathbb{F}_2^3$.

The above definition considers the bit rotation for input, but not for the function output. For generalization of linear approximation with rotation, we include the additional parameter e for circular shift in the output, which is defined as follows.

Definition 7 (Generalized Linear Approximation with Rotations). *Let* $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$. *For* $a \in \mathbb{F}_p^m, b \in \mathbb{F}_p^n, c \in \mathbb{F}_p, d \in \mathbb{F}_{n\log(p)}, e \in \mathbb{F}_{m\log(p)}$, *the generalized linear approximation is the number of solutions satisfying* $a \cdot (f(v) \lll e) = b \cdot (v \lll d) + c$,

$$L^f(a, b, c, d, e) = \#\{v \in \mathbb{F}_p^n | a \cdot (f(v) \lll e) = b \cdot (v \lll d) + c\}.$$

Similarly, for Sbox $T : \mathbb{F}_2^8 \to \mathbb{F}_2^8$, *the linear approximation with rotation is defined as*

$$L^T_{a,b,c,d,e} = \#\{x \in \mathbb{F}_2^8 | a \cdot (T(x) \lll e) = b \cdot (x \lll d) + c\},$$

where $a, b, c \in \mathbb{F}_2^8$ *and* $d, e \in \mathbb{F}_2^3$.

Based on Definitions 6 and 7, the input space of $\mathbb{F}_2^8$ is no longer the full $2^8 = 256$ combinations for Sbox T. As circular shift is involved, any input combinations which are circular shift of another are denoted circularly equivalent. Analogously, output space under the generalized linear approximation with rotation is also much smaller than the original 256 entries in $\mathbb{F}_2^8$. Since both input space and output space are from the same field $\mathbb{F}_2^8$, the number of unique combinations of circularly equivalent vectors is 36. The complete list of such vectors is shown in Table 1, under different binary vector length of $len = 2, \ldots, 8$ for $\mathbb{F}_2^{len}$.

With linear approximation with rotations, we are able to detect the unique characteristic of "011$*$" sequence in Skyscraper-v2 Sbox's input, where $*$ indicates the inclusion of more bit(s) of 1s. Please note that the $x \lll d$ is considered for this "011$*$" as it can potentially wrap around from lower right bits to the upper left bits. If such input sequence does not exist, then output y is simply $y = x \lll 1$. This corresponds to the $2 + 46$ input-output pairs of type $\mathcal{AWD}_0^T$. However, the remaining 16 pairs in $\mathcal{AWD}_0^T$ as well as other types $\mathcal{AWD}_1^T$ and $\mathcal{AWD}_2^T$ have at least one occurrence of "011$*$". Then, the following modeling is required and elaborated below.

First, we start by analyzing the input x with only one such sequence "011$*$". Suppose the starting bit position for this sequence "011$*$" is at kth location, i.e., the bit 0 at position k. Then we can easily identify the corresponding changes to the Sbox output y.

$$y = (x \oplus 1_{(k+1) \bmod 8}) \lll 1,$$

where vector $1_{(k+1) \bmod 8}$ is an one-hot vector bit 1 at position $(k+1)$ mod 8. We can also express it equivalently with circular left shift at the output size, $y \lll 7 = x \oplus 1_{(k+1) \bmod 8}$. For example, input $x = 10101001$ and its corresponding output $y = 01011011$ have one such sequence "011" from bit positions $[1, 0, 7]$ (wrapped) with bit 0 located at position 1; then the one-hot vector 00000100 is applied to the input x to derive the correct output $y = (x \oplus 00000100) \lll 1 = (10101001 + 00000100) \lll 1 = 01011011$. Readers can verify the remaining entries from Table 4.

Second, this input sequence of "011$*$" can occur multiple time inside input vector x. We have experimentally verified that "011$*$" can appear with at most

Table 1: Unique vectors under rotational linearity for different binary vector length of $len = 2, \ldots, 8$. The last row records the total number of unique vectors n_{len} of $\mathbb{F}_2^{len}$.

Vector length len (in bits)						
2	3	4	5	6	7	8
00	000	0000	00000	000000	0000000	00000000
01	001	0001	00001	000001	0000001	00000001
11	011	0011	00011	000011	0000011	00000011
	111	0101	00101	000101	0000101	00000101
		0111	00111	000111	0000111	00000111
		1111	01011	001001	0001001	00001001
			01111	001011	0001011	00001011
			11111	001101	0001101	00001101
				001111	0001111	00001111
				010101	0010011	00010001
				010111	0010101	00010011
				011011	0010111	00010101
				011111	0011011	00010111
				111111	0011101	00011001
					0011111	00011011
					0101011	00011101
					0101111	00011111
					0110111	00100101
					0111111	00100111
					1111111	00101011
						00101101
						00101111
						00110011
						00110101
						00110111
						00111011
						00111101
						00111111
						01010101
						01010111
						01011011
						01011111
						01101111
						01110111
						01111111
						11111111
n_{len} 3	4	6	8	14	20	36

two times in the 8-bit binary vector. Suppose such two sequences of "011$*$" have the starting positions of k and j, respectively. The output y can be written as

$$y = (x \oplus 1_{(k+1) \bmod 8,(j+1) \bmod 8}) \lll 1,$$

where vector $1_{(k+1) \bmod 8,(j+1) \bmod 8}$ is a two-hot vector with bit 1s at positions $(k+1) \bmod 8$ and $(j+1) \bmod 8$.

Lastly, with regard to the 16 input-output pairs of type $\mathcal{AWD}_0^T$, we give the following explanation. Please note that discussion is also applicable to the 8 input-output pairs $y = x \lll 5$ under $\mathcal{AWD}_0^T$. When having two occurrence of "011$*$', one sequence's one-hot vector XORed 1 at the colliding bit position of the other sequence's original bit 1. Equivalently, an additional bit 1 in included while another bit 1 at the different position is removed. For example, input $x =$ 11000110 and its corresponding output $y =$ 10101001 have two such sequence "011" from bit positions $[0, 7, 6]$ and $[3, 2, 1]$. The two-hot vector is 00010010 to determine the output $y = (x \oplus 00010010) \lll 1 = (11000110 \oplus 00010010) \lll 1 =$ 10101001. Therefore, with circular shift embedded in the linear approximation, we are able to better model the Sbox y with respect to its input x.

4 Conclusion and Future Work

The rise of applications with zero-knowledge proofs fuels the proposal for novel arithmetization-oriented hash functions, i.e., `Monolith` and Skyscraper-v2. Such hash functions emphasize on being circuit-friendly, e.g., efficient arithmetization with R1CS constraints. In this paper, we take a closer look at Skyscraper-v2 and analyze its Sbox construction with the inclusion of rotational structure for linear approximations. We provide definitions for linear approximation with rotation and its generalized version. We identify the recurring sequence in Skyscraper-v2 Sbox and how it can be better captured with our analysis. We would like to continue this by including the rotational structure in the Fourier analysis as our future work.

References

1. Albrecht, M.R., Cid, C., Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELlous and MiMC. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 371–397. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34618-8˙13
2. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 191–219. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6˙7

3. Ashur, T., Buschman, T., Mahzoun, M.: Algebraic cryptanalysis of the HADES design strategy: Application to poseidon and Poseidon2. In: Zhu, T., Li, Y. (eds.) ACISP 24, Part II. LNCS, vol. 14896, pp. 225–244. Springer, Singapore (Jul 2024). https://doi.org/10.1007/978-981-97-5028-3˙12
4. Ashur, T., Mahzoun, M., Posen, J., Šijačić, D.: Vision mark-32: ZK-friendly hash function over binary tower fields. Cryptology ePrint Archive, Report 2024/633 (2024), https://eprint.iacr.org/2024/633
5. Bak, A.: A practical distinguisher on the full skyscraper permutation. Cryptology ePrint Archive, Report 2025/102 (2025), https://eprint.iacr.org/2025/102
6. Bak, A., Jazeron, G., Galissant, P., Perrin, L.: Attacking split-and-lookup-based primitives using probabilistic polynomial system solving applications to round-reduced monolith and full-round skyscraper. IACR Trans. Symm. Cryptol. **2025**(3), 337–367 (2025). https://doi.org/10.46586/tosc.v2025.i3.337-367
7. Bak, A., Perrin, L.: On the security of split-and-lookup-based ZK-friendly primitives. IACR Trans. Symm. Cryptol. **2025**(2), 87–123 (2025). https://doi.org/10.46586/tosc.v2025.i2.87-123
8. Bariant, A., Boeuf, A., Lemoine, A., Ayala, I.M., Øygarden, M., Perrin, L., Raddum, H.: The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part IV. LNCS, vol. 14923, pp. 139–173. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68385-5˙5
9. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 2–21. Springer, Berlin, Heidelberg (Aug 1991). https://doi.org/10.1007/3-540-38424-3˙1
10. Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 487–496. Springer, Berlin, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4˙34
11. Bouvier, C., Briaud, P., Chaidos, P., Perrin, L., Salen, R., Velichkov, V., Willems, D.: New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and Jive compression mode. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part III. LNCS, vol. 14083, pp. 507–539. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38548-3˙17
12. Bouvier, C., Grassi, L., Khovratovich, D., Koschatko, K., Rechberger, C., Schmid, F., Schofnegger, M.: Skyscraper: Fast hashing on big primes. IACR TCHES **2025**(2), 743–780 (2025). https://doi.org/10.46586/tches.v2025.i2.743-780
13. Bouvier, C., Grassi, L., Khovratovich, D., Koschatko, K., Rechberger, C., Schmid, F., Schofnegger, M.: Skyscraper-v2: Fast hashing on big primes. Cryptology ePrint Archive, Report 2025/058 (2025), https://eprint.iacr.org/2025/058
14. Daemen, J.: Cipher and hash function design strategies based on linear and differential cryptanalysis. doctoral dissertation, k.u. leuven, 1995. (1995), https://cs.ru.nl/~joan/papers/JDA_Thesis_1995.pdf
15. Eichlseder, M., Grassi, L., Lüftenegger, R., Øygarden, M., Rechberger, C., Schofnegger, M., Wang, Q.: An algebraic attack on ciphers with low-degree round functions: Application to full MiMC. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 477–506. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64837-4˙16
16. Grassi, L., Khovratovich, D., Koschatko, K., Rechberger, C., Schofnegger, M., Schröppel, V., Wu, Z.: Poseidon(2)b: Binary field versions of poseidon/Poseidon2. CiC **2**(4), 15 (2026). https://doi.org/10.62056/a66ce0zn4

17. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Reinforced concrete: A fast hash function for verifiable computation. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 1323–1335. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560686
18. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Monolith: Circuit-friendly hash functions with new nonlinear layers for fast and constant-time implementations. IACR Trans. Symm. Cryptol. **2024**(3), 44–83 (2024). https://doi.org/10.46586/tosc.v2024.i3.44-83
19. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 519–535. USENIX Association (Aug 2021), `https://www.usenix.org/conference/usenixsecurity21/presentation/grassi`
20. Grassi, L., Khovratovich, D., Schofnegger, M.: Poseidon2: A faster version of the poseidon hash function. In: El Mrabet, N., De Feo, L., Duquesne, S. (eds.) AFRICACRYPT 23. LNCS, vol. 14064, pp. 177–203. Springer, Cham (Jul 2023). https://doi.org/10.1007/978-3-031-37679-5˙8
21. Grassi, L., Koschatko, K., Rechberger, C.: Poseidon and neptune: Gröbner basis cryptanalysis exploiting subspace trails. IACR Trans. Symm. Cryptol. **2025**(2), 34–86 (2025). https://doi.org/10.46586/tosc.v2025.i2.34-86
22. Grassi, L., Onofri, S., Pedicini, M., Sozzi, L.: Invertible quadratic non-linear layers for MPC-/FHE-/ZK-friendly schemes over $\mathbb{F}_p^n$: Application to Poseidon. IACR Trans. Symm. Cryptol. **2022**(3), 20–72 (2022). https://doi.org/10.46586/tosc.v2022.i3.20-72
23. Koschatko, K., Lüftenegger, R., Rechberger, C.: Exploring the six worlds of Gröbner basis cryptanalysis: Application to anemoi. IACR Trans. Symm. Cryptol. **2024**(4), 138–190 (2024). https://doi.org/10.46586/tosc.v2024.i4.138-190
24. Liu, F., Koschatko, K., Grassi, L., Yan, H., Chen, S., Banik, S., Meier, W.: Opening the blackbox: Collision attacks on round-reduced Tip5, Tip4, Tip4' and monolith. IACR Trans. Symm. Cryptol. **2024**(4), 97–137 (2024). https://doi.org/10.46586/tosc.v2024.i4.97-137
25. Matsui, M.: The first experimental cryptanalysis of the data encryption standard. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 1–11. Springer, Berlin, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5˙1
26. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 386–397. Springer, Berlin, Heidelberg (May 1994). https://doi.org/10.1007/3-540-48285-7˙33
27. Sanso, A., Vitto, G.: Attacking poseidon via graeffe-based root-finding over NTT-friendly fields. Cryptology ePrint Archive, Report 2025/937 (2025), `https://eprint.iacr.org/2025/937`
28. Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive, Report 2020/1143 (2020), `https://eprint.iacr.org/2020/1143`
29. Szepieniec, A., Lemmens, A., Sauer, J.F., Threadbare, B.: The Tip5 hash function for recursive STARKs. Cryptology ePrint Archive, Report 2023/107 (2023), `https://eprint.iacr.org/2023/107`
30. Zhao, Z., Ding, J.: Breaking poseidon challenges with graeffe transforms and complexity analysis by FFT lower bounds. Cryptology ePrint Archive, Report 2025/950 (2025), `https://eprint.iacr.org/2025/950`

A Sbox Design in Skyscraper-v2

Table 2: Bitwise Computation inside Skyscraper-v2 SBox without the last bit rotation $\lll 1$.

variable	Bit Position							
	7	6	5	4	3	2	1	0
$x \oplus$	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
$(\overline{x} \lll 1$	$\overline{x_6}$	$\overline{x_5}$	$\overline{x_4}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_0}$	$\overline{x_7}$
$\odot x \lll 2$	x_5	x_4	x_3	x_2	x_1	x_0	x_7	x_6
$\odot x \lll 3)$	x_4	x_3	x_2	x_1	x_0	x_7	x_6	x_5
u	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0

Table 3: Skyscraper-v2 SBox bit-decomposed result $y = (y_7 \dots y_0) = T(x)$.

$$
\begin{aligned}
x_7 = u_6 &= (\overline{x_5} \cdot x_4 \cdot x_3) \oplus x_6 \\
x_6 = u_5 &= (\overline{x_4} \cdot x_3 \cdot x_2) \oplus x_5 \\
x_5 = u_4 &= (\overline{x_3} \cdot x_2 \cdot x_1) \oplus x_4 \\
x_4 = u_3 &= (\overline{x_2} \cdot x_1 \cdot x_0) \oplus x_3 \\
x_3 = u_2 &= (\overline{x_1} \cdot x_0 \cdot x_7) \oplus x_2 \\
x_2 = u_1 &= (\overline{x_0} \cdot x_7 \cdot x_6) \oplus x_1 \\
x_1 = u_0 &= (\overline{x_7} \cdot x_6 \cdot x_5) \oplus x_0 \\
x_0 = u_7 &= (\overline{x_6} \cdot x_5 \cdot x_4) \oplus x_7
\end{aligned}
$$

Table 4: Skyscraper-v2 Sbox input-output pairs (x, y), with $x, y \in \mathbb{F}_{2^8}, y = T(x) = (x \oplus ((\overline{x} \lll 1) \odot (x \lll 2))) \lll 1$.

x = 00000000, y = 00000000	x = 01000000, y = 10000000	x = 10000000, y = 00000001	x = 11000000, y = 10000101
x = 00000001, y = 00000010	x = 01000001, y = 10000010	x = 10000001, y = 00001011	x = 11000001, y = 10001011
x = 00000010, y = 00000100	x = 01000010, y = 10000100	x = 10000010, y = 00000101	x = 11000010, y = 10000001
x = 00000011, y = 00010110	x = 01000011, y = 10010110	x = 10000011, y = 00010111	x = 11000011, y = 10010111
x = 00000100, y = 00001000	x = 01000100, y = 10001000	x = 10000100, y = 00001001	x = 11000100, y = 10001101
x = 00000101, y = 00001010	x = 01000101, y = 10001010	x = 10000101, y = 00000011	x = 11000101, y = 10000011
x = 00000110, y = 00101100	x = 01000110, y = 10101100	x = 10000110, y = 00101101	x = 11000110, y = 10101001
x = 00000111, y = 00101110	x = 01000111, y = 10101110	x = 10000111, y = 00101111	x = 11000111, y = 10101111
x = 00001000, y = 00010000	x = 01001000, y = 10010000	x = 10001000, y = 00010001	x = 11001000, y = 10010101
x = 00001001, y = 00010010	x = 01001001, y = 10010010	x = 10001001, y = 00011011	x = 11001001, y = 10011011
x = 00001010, y = 00010100	x = 01001010, y = 10010100	x = 10001010, y = 00010101	x = 11001010, y = 10010001
x = 00001011, y = 00000110	x = 01001011, y = 10000110	x = 10001011, y = 00000111	x = 11001011, y = 10000111
x = 00001100, y = 01011000	x = 01001100, y = 11011000	x = 10001100, y = 01011001	x = 11001100, y = 11011101
x = 00001101, y = 01011010	x = 01001101, y = 11011010	x = 10001101, y = 01010011	x = 11001101, y = 11010011
x = 00001110, y = 01011100	x = 01001110, y = 11011100	x = 10001110, y = 01011101	x = 11001110, y = 11011001
x = 00001111, y = 01011110	x = 01001111, y = 11011110	x = 10001111, y = 01011111	x = 11001111, y = 11011111
x = 00010000, y = 00100000	x = 01010000, y = 10100000	x = 10010000, y = 00100001	x = 11010000, y = 10100101
x = 00010001, y = 00100010	x = 01010001, y = 10100010	x = 10010001, y = 00101011	x = 11010001, y = 10101011
x = 00010010, y = 00100100	x = 01010010, y = 10100100	x = 10010010, y = 00100101	x = 11010010, y = 10100001
x = 00010011, y = 00110110	x = 01010011, y = 10110110	x = 10010011, y = 00110111	x = 11010011, y = 10110111
x = 00010100, y = 00101000	x = 01010100, y = 10101000	x = 10010100, y = 00101001	x = 11010100, y = 10101101
x = 00010101, y = 00101010	x = 01010101, y = 10101010	x = 10010101, y = 00100011	x = 11010101, y = 10100011
x = 00010110, y = 00001100	x = 01010110, y = 10001100	x = 10010110, y = 00001101	x = 11010110, y = 10001001
x = 00010111, y = 00001110	x = 01010111, y = 10001110	x = 10010111, y = 00001111	x = 11010111, y = 10001111
x = 00011000, y = 10110000	x = 01011000, y = 00110000	x = 10011000, y = 10110001	x = 11011000, y = 00110101
x = 00011001, y = 10110010	x = 01011001, y = 00110010	x = 10011001, y = 10111011	x = 11011001, y = 00111011
x = 00011010, y = 10110100	x = 01011010, y = 00110100	x = 10011010, y = 10110101	x = 11011010, y = 00110001
x = 00011011, y = 10100110	x = 01011011, y = 00100110	x = 10011011, y = 10100111	x = 11011011, y = 00100111
x = 00011100, y = 10111000	x = 01011100, y = 00111000	x = 10011100, y = 10111001	x = 11011100, y = 00111101
x = 00011101, y = 10111010	x = 01011101, y = 00111010	x = 10011101, y = 10110011	x = 11011101, y = 00110011
x = 00011110, y = 10111100	x = 01011110, y = 00111100	x = 10011110, y = 10111101	x = 11011110, y = 00111001
x = 00011111, y = 10111110	x = 01011111, y = 00111110	x = 10011111, y = 10111111	x = 11011111, y = 00111111
x = 00100000, y = 01000000	x = 01100000, y = 11000010	x = 10100000, y = 01000001	x = 11100000, y = 11000101
x = 00100001, y = 01000010	x = 01100001, y = 11000000	x = 10100001, y = 01001011	x = 11100001, y = 11001011
x = 00100010, y = 01000100	x = 01100010, y = 11000110	x = 10100010, y = 01000101	x = 11100010, y = 11000001
x = 00100011, y = 01010110	x = 01100011, y = 11010100	x = 10100011, y = 01010111	x = 11100011, y = 11010111
x = 00100100, y = 01001000	x = 01100100, y = 11001010	x = 10100100, y = 01001001	x = 11100100, y = 11001101
x = 00100101, y = 01001010	x = 01100101, y = 11001000	x = 10100101, y = 01000011	x = 11100101, y = 11000011
x = 00100110, y = 01101100	x = 01100110, y = 11101110	x = 10100110, y = 01101101	x = 11100110, y = 11101001
x = 00100111, y = 01101110	x = 01100111, y = 11101100	x = 10100111, y = 01101111	x = 11100111, y = 11101111
x = 00101000, y = 01010000	x = 01101000, y = 11010010	x = 10101000, y = 01010001	x = 11101000, y = 11010101
x = 00101001, y = 01010010	x = 01101001, y = 11010000	x = 10101001, y = 01011011	x = 11101001, y = 11011011
x = 00101010, y = 01010100	x = 01101010, y = 11010110	x = 10101010, y = 01010101	x = 11101010, y = 11010001
x = 00101011, y = 01000110	x = 01101011, y = 11000100	x = 10101011, y = 01000111	x = 11101011, y = 11000111
x = 00101100, y = 00011000	x = 01101100, y = 10011010	x = 10101100, y = 00011001	x = 11101100, y = 10011101
x = 00101101, y = 00011010	x = 01101101, y = 10011000	x = 10101101, y = 00010011	x = 11101101, y = 10010011
x = 00101110, y = 00011100	x = 01101110, y = 10011110	x = 10101110, y = 00011101	x = 11101110, y = 10011001
x = 00101111, y = 00011110	x = 01101111, y = 10011100	x = 10101111, y = 00011111	x = 11101111, y = 10011111
x = 00110000, y = 01100001	x = 01110000, y = 11100010	x = 10110000, y = 01100000	x = 11110000, y = 11100101
x = 00110001, y = 01100011	x = 01110001, y = 11100000	x = 10110001, y = 01101010	x = 11110001, y = 11101011
x = 00110010, y = 01100101	x = 01110010, y = 11100110	x = 10110010, y = 01100100	x = 11110010, y = 11100001
x = 00110011, y = 01110111	x = 01110011, y = 11110100	x = 10110011, y = 01110110	x = 11110011, y = 11110111
x = 00110100, y = 01101001	x = 01110100, y = 11101010	x = 10110100, y = 01101000	x = 11110100, y = 11101101
x = 00110101, y = 01101011	x = 01110101, y = 11101000	x = 10110101, y = 01100010	x = 11110101, y = 11100011
x = 00110110, y = 01001101	x = 01110110, y = 11001110	x = 10110110, y = 01001100	x = 11110110, y = 11001001
x = 00110111, y = 01001111	x = 01110111, y = 11001100	x = 10110111, y = 01001110	x = 11110111, y = 11001111
x = 00111000, y = 01110001	x = 01111000, y = 11110010	x = 10111000, y = 01110000	x = 11111000, y = 11110101
x = 00111001, y = 01110011	x = 01111001, y = 11110000	x = 10111001, y = 01111010	x = 11111001, y = 11111011
x = 00111010, y = 01110101	x = 01111010, y = 11110110	x = 10111010, y = 01110100	x = 11111010, y = 11110001
x = 00111011, y = 01100111	x = 01111011, y = 11100100	x = 10111011, y = 01100110	x = 11111011, y = 11100111
x = 00111100, y = 01111001	x = 01111100, y = 11111010	x = 10111100, y = 01111000	x = 11111100, y = 11111101
x = 00111101, y = 01111011	x = 01111101, y = 11111000	x = 10111101, y = 01110010	x = 11111101, y = 11110011
x = 00111110, y = 01111101	x = 01111110, y = 11111110	x = 10111110, y = 01111100	x = 11111110, y = 11111001
x = 00111111, y = 01111111	x = 01111111, y = 11111100	x = 10111111, y = 01111110	x = 11111111, y = 11111111