# Practical Collision Attacks on Reduced-Round Xoodyak Hash Mode

Huina Li[1], Le He[2], and Weidong Qiu[1]✉

[1] School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2] School of Cyber Engineering, Xidian University, Xi'an, China
{lihuina, qiuwd}@sjtu.edu.cn

**Abstract.** Xoodyak is a finalist of the NIST lightweight cryptography competition, offering both keyed and hash modes. After several years of cryptanalysis, the largest number of Xoodyak hash rounds for which actual collisions was still in vacancy. To the best of our knowledge, one of the most powerful collision attacks on hash functions based on sponge construction is the differential-based attacks using the S-box linearization technique proposed by Qiao *et al.* (EUROCRYPT 2017). However, the linearization technique requires a large number of degrees of freedom, making it challenging to apply to Xoodyak with a small outer part. On the other hand, the constraint-input and constraint-output imposed on the differential trail of Xoodoo permutation make the exhaustive search for such high-probability differential trails in collision attacks extremely costly.

In this paper, we present critical observations regarding Xoodoo round function, particularly focusing on its unique $\theta$ and $\chi$ operation. These properties can be leveraged to manually design specific differential trails for the Xoodoo permutation, referred to as *loop* differential trails. To efficiently find practical collisions for up to 3 rounds, we develop a SAT model based on these *loop* trails. Finally, we present the first practical collision on 2 rounds and a practical semi-free-start collision on 3 rounds of Xoodyak hash mode. Besides, we improve Dong *et al.*'s (CRYPTO 2024) collision attack on 3-round Xoodyak-Hash from $2^{125.23}$ to $2^{100.93}$ using several linearization strategies. Since we focus on the analysis on collisions during the message absorbing phase of the hash modes, our results are applicable to both Xoodyak-Hash and Xoodyak-Xof.

**Keywords:** Xoodyak· Xoodyak hash mode · Collision attack · Semi-free-start collision attack

## 1 Introduction

With the rapid development of emerging networks such as the Internet of Things (IoT), sensor networks, and vehicular networks, symmetric cryptographic algorithms face new challenges. In 2018, the National Institute of Standards and Technology (NIST) launched a Lightweight Cryptography Competition[18] to solicit, evaluate, and standardize schemes providing authenticated encryption with

associated data (AEAD) and optional hashing functionalities for constrained environments. In April 2019, NIST announced the first round of 56 candidate algorithms. Based on extensive security evaluations of the candidates by cryptanalysts, NIST announced the 10 finalists for the final round in March 2021. Xoodyak [6] is one of these finalists and has attracted considerable attention from cryptographers due to its unique design.

Xoodyak offers both the keyed and the hash modes, while the keyed mode has been extensively studied [22,27], the hash mode has received comparatively less attention [9]. In this paper, we focus on the Xoodyak hash mode. The Xoodyak hash mode is a versatile cryptographic primitive that combines sponge construction [10] with the Xoodoo permutation [5], operating under a mode of operation known as Cyclist. Xoodyak hash mode includes the hash function Xoodyak-Hash and an eXtendable Output Function (XOF) Xoodyak-Xof. Both of them have an internal state size of $b = 384$ bits and an output size $h$ of either 256, or an arbitrary output length, respectively. Xoodyak is built on the sponge construction, the underlying permutation Xoodoo consists of 12 rounds of a non-linear round function applied to the 384-bit state.

A successful collision attack on a sponge-based hash function $H$ is to find a pair of distinct messages $M$ and $M'$ such that $H_{IV}(M) = H_{IV}(M')$ with a time complexity of less than $2^{min(h/2,c/2)}$. To the best of our knowledge, the most powerful collision attack on sponge-based hash function to date is based on the differential attack framework proposed by Dinur *et al.* in [8]. The full attack primarily consists of two parts: the first part is the high probability differential trail search that ensures collision on the digest bits, which we refer to as the colliding trail search phase, and the second part is constructing a 1-round connector with the target difference algorithm (TDA) in order to obtain a sufficiently large set of message pairs that simultaneously satisfy the colliding trail, as well as the constraints imposed by the padding rule and the initial value (IV) of the hash function. Qiao *et al.* [19] built upon the framework proposed by Dinur *et al.* [8] and introduced a novel algebraic technique to linearize all S-boxes in the first round, thereby extending 1-round connectors to 2-round connectors. Furthermore, 3-round connectors were designed in [23,11] using partial linearization techniques, resulting in the first practical 5-round collisions for SHA3-224 and SHA3-256.

To further address the inefficiency of differential trail search, Guo et al. in [12] introduced the SAT tool to accelerate multiple intermediate steps of the collision attacks. This approach demonstrated remarkably high efficiency in solving both differential trail search and connecting trail search problems encountered during the process, ultimately resulting in the first six-round collision attack on SHAKE128. Huang *et al.* [13] observed that the previously proposed linearization techniques [19,23,11] become impractical for variants with smaller input spaces, such as SHA3-384, which has only 832 initial degrees of freedom (ignoring padding bits). To address this limitation, they proposed an attack framework that utilizes 2-block messages instead of 1-block messages and transforms the connectivity problem into a Boolean Satisfiability (SAT) problem. Using SAT-

based tools, they successfully presented the first practical attack on 4-round SHA3-384.

The Constrained-Input Constrained-Output (CICO) problem [10] inherent to sponge-based hash functions, significantly restricts the availability of high-probability differential trails for collision attacks. In the case of the Xoodyak hash mode, the rate part is considerably smaller, with only the first 128 bits having non-zero differences. Such trails are often extremely rare, requiring an exhaustive search of the entire space, which is computationally expensive. So far, differential trail search results on the Xoodoo permutation have only been reported by its designers [5,17], who developed a dedicated tool for general trail search. This may explain why the first collision results for Xoodyak, relying on a meet-in-the-middle (MitM) attack, were recently presented in 2024, rather than through a differential-based attack.

At CRYPTO 2024, Dong *et al.* [9] proposed a generic MitM collision attack framework for sponge constructions for the first time and presented the first collision attack on 3-round Xoodyak-Hash with the time complexity of $2^{125.23}$. However, it required a huge memory complexity of $2^{124}$.

### 1.1 Our Contributions.

In this work, we improve the differential-based collision attacks against Xoodyak hash mode, as summarized below:

- We adopt a multi-block collision attack framework for the reduced-round Xoodyak hash mode. Unlike previous works that focused on the squeezing phase of Keccak [8,19,23,11,13,12], our attack focuses on collisions during the message absorbing phase of hash modes. This distinction makes our framework applicable to both the hash mode (Xoodyak-Hash) and the extendable output function (Xoodyak-Xof).
- We utilize the property of the $\theta$ operation: when all columns in a sheet are odd, then the $\theta$-effect cancels and there are no induced affected columns. Leveraging this property, along with the differential property of the non-linear operation $\chi$, we manually design special differential trails (we refer to them as *loop* differential trails), each maintaining a probability of $2^{-64}$ per round. To efficiently find practical collisions, we develop a collision search SAT model based on the *loop* trails. As a result, we obtain an actual 2-round collision in 64.7 seconds, and a practical 3-round semi-free-start (SFS) collision in 3.67 seconds. These are the first practical collision results on reduced rounds of the Xoodyak hash mode. For the source code, please refer to `https://github.com/HuinaLi/Practical-Collision-Attacks-on-Xoodyak`.
- Based on 3-round *loop* differential trail with the probability of $2^{-192}$, we propose a specific linearization technique that carefully selects the output bits of the first round to be linearized. We decrease the time complexity of Dong *et al.*'s 3-round collision attacks on Xoodyak-Hash from $2^{125.23}$ to $2^{100.9311}$. It is worth noting that our approach has negligible memory complexity. The results are summarized in Table 1.

Table 1: Summary of collision attacks on Xoodyak hash mode

| Target | Methods | Rounds | Time | Memory | Reference |
|---|---|---|---|---|---|
| Xoodyak-Hash | MitM | 3 | $2^{125.23}$ | $2^{124}$ | [9] |
| Xoodyak hash mode | Differential | 2 | Practical | Negligible | Section 5 |
| | Differential | 3 | $2^{100.9311}$ | Negligible | Section 5 |

### 1.2 Organization

In Section 2, we define some notations and properties that will be used in the paper and briefly describe Xoodyak hash mode and the underlying permutation Xoodoo. In Section 3, we introduce our collision attack framework. In Section 4, we present a efficient SAT-based collision search model. In Section 5 we give some new collision results on Xoodyak hash mode. Finally, we summarize the full paper in Section 6.

## 2 Preliminary

### 2.1 Xoodyak Hash Mode

Xoodyak [6] is a versatile cryptographic primitive combining sponge construction and Xoodoo permutation denoted by $f$, based on an operational mode termed Cyclist. It offers two modes including the keyed and the hash modes, one of which is selected upon initialization. In the case of hash mode illustrated in Figure 1, it includes Xoodyak-Hash and the extanded output function Xoodyak-Xof.

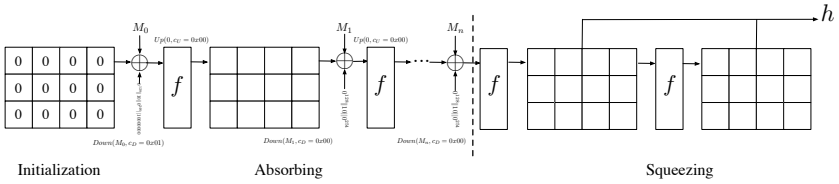


Fig. 1: The Xoodyak Hash Mode

The 384-bit state is initialized to the all-zero string like Cyclist, and a duplexing call corresponds to a sequence of Down() and Up()(Due to page limitations, specific details have been omitted. For more details, please refer to [6]). For Xoodyak hash mode with the parameters $b = 384, r = 130, c = 254$, the security claims are $2^{128}$ against any attacks.

The rate is set to $r = 130$ bits, which includes the maximum input message block size of 128 bits along with 2 bits from the padding. The sponge construction works on a $b$-bit internal state, which is divided into two phases (see Figure 1):

in the absorbing phase, it cuts the input message $M$ in blocks of given length (after padding $r = 130$) and calls Down() and Up() appropriately. The Down($M_i$, $c_D = 0x01$ if first block else $0x00$) method absorbs one block of input message, by bitwise adding it to the state, together with a color $c_D$ in the last byte of the state. The processed message is denoted by $\bar{M}_i$. The Up($0, c_U = 0x00$) method aims at producing one block of output. It absorbs a color $c_U$ in the last byte of the state, invokes the underlying permutation $f$ and returns the output of the state. After absorbing all the message blocks, a 128-bit hash value is extracted during the squeezing process each time, until the desired hash length $h$ is achieved.

### 2.2 Description of Xoodoo

Xoodoo is the underlying permutation of Xoodyak, first presented by Daemen et al. [5] at ToSC 2018. It operates on a 3-dimensional array of size 384 bits. The bits of the state $S$ are indexed by the $(x, y, z)$ coordinates where $0 \leq x < 4$, $0 \leq y < 3$, and $0 \leq z < 32$. Each bit is indexed by $S[32 \times (x + 4 \times y) + z]$. The state can be broken down into rows, columns, and lanes, which refer to the 1-dimensional arrays in the $x$, $y$, and $z$ directions, or into two-dimensional arrays, such as planes and sheets, in the $(x, z)$ and $(y, z)$ directions, as shown in Figure 2.

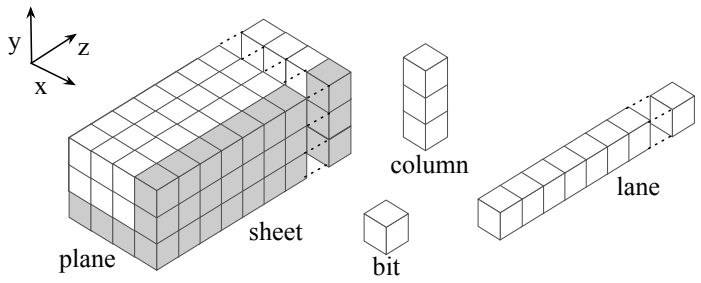


Fig. 2: The Xoodoo state

Xoodoo has an iterated structure, and the input state is processed through the round function for multiple rounds to obtain the output state, with a total of 12 rounds. The round function of Xoodoo follows a design approach similar to Keccak-$p$ with five step mappings in each round: $R = \rho_{east} \circ \chi \circ \iota \circ \rho_{west} \circ \theta$. The five step mappings in each round of the permutation are given below, in order:
$\theta$ creates a parity plane $P$ by adding the bit-wise sum of all columns, if the parity of a column is 1, which defines an odd (*resp.* even) column. Then, the $\theta$-effect of a state value $S$ is $E$, which is calculated by adding two shifted parity planes with shift offsets $(1, 5)$ and $(1, 14)$. The $\theta$-effect is then added to the original input

state.

$$\begin{aligned} P[x][z] &\leftarrow \sum_{y=0}^{2} S[x][y][z] \\ E[x][z] &\leftarrow P[x+1][z+5] \oplus P[x+1][z+14] \\ S[x][y][z] &\leftarrow S[x][y][z] \oplus E[x][z] \end{aligned} \tag{1}$$

$\rho_{west}$ shifts two planes of the state with offsets $(1, 0)$ and $(0, 11)$, respectively.

$$\begin{aligned} S[x][1][z] &\leftarrow S[x+1][1][z] \\ S[x][2][z] &\leftarrow S[x][2][z+11] \end{aligned} \tag{2}$$

$\iota$ adds a constant in each round. The round constants $C_i$ in hexadecimal notation (see Table 2) are such that the least significant bit is at $z = 0$.

$$S[0][0][z] \leftarrow S[0][0][z] \oplus C_i$$

Table 2: Constants $C_i$ used in the Xoodoo (round index $i$ from $-11$ to 0)

| Round $i$ | Constant $C_i$ | Round $i$ | Constant $C_i$ | Round $i$ | Constant $C_i$ |
|---|---|---|---|---|---|
| -11 | 0x00000058 | -7 | 0x00000120 | -3 | 0x00000380 |
| -10 | 0x00000038 | -6 | 0x00000014 | -2 | 0x000000f0 |
| -9 | 0x000003c0 | -5 | 0x00000060 | -1 | 0x000001a0 |
| -8 | 0x000000d0 | -4 | 0x0000002c | 0 | 0x00000012 |

$\chi$ is the only non-linear layer. It operates in parallel on 3-bit columns and, as such, forms a layer of 128 3-bit S-boxes.

$$\begin{aligned} S[x][0][z] &\leftarrow S[x][0][z] \oplus \neg S[x][1][z] \wedge S[x][2][z] \\ S[x][1][z] &\leftarrow S[x][1][z] \oplus \neg S[x][2][z] \wedge S[x][0][z] \\ S[x][2][z] &\leftarrow S[x][2][z] \oplus \neg S[x][0][z] \wedge S[x][1][z] \end{aligned} \tag{3}$$

$\rho_{east}$ shifts two planes of the state with offsets $(0, 1)$ and $(2, 8)$, respectively.

$$\begin{aligned} S[x][1][z] &\leftarrow S[x][1][z+1] \\ S[x][2][z] &\leftarrow S[x+2][2][z+8] \end{aligned} \tag{4}$$

### 2.3 Properties of $\theta$

We first present the definition of *loop*, an important property of $\theta$ that allows us to manually design differential trails in Section 3.3 for enhanced collision attacks.

**Definition 1 (Loop [5]).** *A parity loop (or loop for short) is a state value with 32 active bits in a sheet, each in a distinct column.*

A *loop* is invariant through $\theta$. According to Equation 1, a *loop* generates 32 odd columns, that means all the columns in a sheet are odd, *i.e.*, $P[x+1][z+5] \oplus P[x+1][z+14] = 0$, then the $\theta$-effect cancels and there are no induced affected columns.

It is worth noting that, as a linear function, the properties of $\theta$ remain the same whether it is applied to the absolute value of a state or to a difference. For a $n$-round differential trail, if all the internal differences consist solely of *loop* elements, we refer to such a differential trail as a *loop differential trail*. More details for constructing 2-round or 3-round *loop differential trails* will be introduced in Section 3.3.

### 2.4 Properties of $\chi$

$\chi$ is a key non-linear component in the round function of Xoodoo for generating differential probability, which leading certain input and output difference to appear in a non-random way. It is applied to each column of the state independently, and can be regarded as a 3-bit S-box. We first present the difference distribution table (DDT) for the 3-bit S-box (see Table 3)

**Definition 2 (Valid Differential Pattern).** *In the DDT of Xoodoo's 3-bit S-box, if the entry $\sigma(\alpha, \beta)$ for the input-output differential pair $(\alpha, \beta)$ is non-zero, we call the differential pattern valid; otherwise, it is referred to as invalid.*

**Definition 3 (Active S-box).** *A S-box is active if its input difference is non-zero; conversely, it is termed inactive if $(\alpha, \beta) = (0, 0)$.*

Each valid difference pattern $(\alpha, \beta)$ has a non-zero value, *i.e.*, the entry $\sigma(\alpha, \beta) \neq 0$. The differential weight of each valid pattern can be denoted by $w_{(\alpha,\beta)} = -log_2 \frac{\sigma(\alpha,\beta)}{8} = 2$.

Table 3: DDT of Xoodoo's S-box

| Input difference ($\alpha$) | Output difference ($\beta$) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0x0 | 0x1 | 0x2 | 0x3 | 0x4 | 0x5 | 0x6 | 0x7 |
| 0x0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x1 | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 2 |
| 0x2 | 0 | 0 | 2 | 2 | 0 | 0 | 2 | 2 |
| 0x3 | 0 | 2 | 2 | 0 | 0 | 2 | 2 | 0 |
| 0x4 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 |
| 0x5 | 0 | 2 | 0 | 2 | 2 | 0 | 2 | 0 |
| 0x6 | 0 | 0 | 2 | 2 | 2 | 2 | 0 | 0 |
| 0x7 | 0 | 2 | 2 | 0 | 2 | 0 | 0 | 2 |

*Property 1.* Given an input difference $\alpha \in \mathbb{F}_2^3$ and an output difference $\beta \in \mathbb{F}_2^3$ such that $\sigma(\alpha, \beta) \neq 0$, denote the value solution set $X = \{x \in \mathbb{F}_2^3 | \chi(x) \oplus \chi(x \oplus \alpha) = \beta\}$ and $\chi(X) = \{\chi(x) : x \in X\}$, then $X$ is a linearizable affine subspace.

The algebraic degree of the 3-bit XOODOO's S-box is 2 as shown in Equation 5, which means that for any valid differential pattern $(\alpha, \beta)$, the set of input messages $x$ that follow this pattern forming an affine subspace of size $2^{3-w_{(\alpha,\beta)}}$.

**3-bit S-box Linearization.** The linearization of a 3-bit S-box can be categorized into two classes.

- For active S-boxes, according to Property 1, when the input difference and the compatible output difference are given, two linear equations on the input bits $x_i$ can be listed for each active S-box, such that the outputs $y$ can be expressed as linear combinations of the input bits. From this algebraic perspective, these S-boxes are already fully linearized.
- For non-active S-boxes, linearization can be achieved by introducing two additional equations involving the input bits (see Property 2).

  *Property 2.* The algebraic normal form of $\chi$ mapping 3-bit $x = (x_0, x_1, x_2)$ into 3-bit $y = (y_0, y_1, y_2)$ can be written as $y_i = x_i \oplus (x_{i+1} \oplus 1)x_{i+2}$ as below,

$$\begin{aligned} y_0 &= x_0 \oplus (x_1 \oplus 1)x_2 \\ y_1 &= x_1 \oplus (x_2 \oplus 1)x_0 \\ y_2 &= x_2 \oplus (x_0 \oplus 1)x_1 \end{aligned} \tag{5}$$

  Given the 3-bit input $x = (x_0, x_1, x_2)$, two bits of the output $y_{i+1}, y_{i+2}$ can be linearized by introducing one linear equation on $x_i$. Furthermore, any bit of the output $y$ can be represented as a linear function by adding two extra linear equations on the input $x$.

### 2.5 Difference-Value Relations

At CRYPTO 2021, Liu *et al.* [15] proposed an algebraic perspective on differential-linear cryptanalysis. This novel algebraic perspective pointed out that the output difference of a Boolean function is a special Boolean function of the input difference and input value. It is important to note that this forms the theoretical foundation for constructing the difference-value relations model, which is presented in detail in Section 4.3.

*Property 3.* For a Boolean function $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ representing a certain output bit of a primitive, the output difference $\beta_i \in \mathbb{F}_2$ is the derivative of $f$ with respect to input difference $\alpha$ at point $x \in \mathbb{F}_2^n$,

$$\beta_i = \mathcal{D}_\alpha f_i(x) = f_i(x) \oplus f_i(x \oplus \alpha) \tag{6}$$

This is equivalently expressed as the partial derivative of $f_i(x \oplus u \cdot \alpha)$ with respect to the boolean variable $u \in \mathbb{F}_2$. Formally, this relationship is defined as:

$$\mathcal{D}_\alpha f_i(x) = \mathcal{D}_u f_i(x \oplus u \cdot \alpha) \tag{7}$$

where the input message value $x$, the input difference $\alpha$ ($x, \alpha \in \mathbb{F}_2^n$), $u$ is an auxiliary binary variable, $\mathcal{D}_u f_i(x \oplus u \cdot \alpha)$ is the partial derivative of $f_i(x \oplus u \cdot \alpha)$ with respect to $u$.

According to Equation 3, the algebraic normal forms (ANFs) of the output bits are denoted by $\chi_i(x_i, x_{i+1}, x_{i+2}) = x_i \oplus \neg x_{i+1} \cdot x_{i+2}$. Based on Equation 7, the relationship between the value and the differential can be illustrated as follows.

$$\begin{cases} \beta_0 = \mathcal{D}_u \chi_0(x_0 \oplus u \cdot \alpha_0, x_1 \oplus u \cdot \alpha_1, x_2 \oplus u \cdot \alpha_2) \\ \beta_1 = \mathcal{D}_u \chi_1(x_0 \oplus u \cdot \alpha_0, x_1 \oplus u \cdot \alpha_1, x_2 \oplus u \cdot \alpha_2) \\ \beta_2 = \mathcal{D}_u \chi_2(x_0 \oplus u \cdot \alpha_0, x_1 \oplus u \cdot \alpha_1, x_2 \oplus u \cdot \alpha_2) \end{cases} \tag{8}$$

Following the above Boolean equations, when the valid differential pattern $(\alpha, \beta)$ is determined, the linear equations on input bits $x_i$ are directly listed.

*Example 1.* Let us consider an active S-box represented by $S(x_0, x_1, x_2)$, which follows a valid difference pattern $(\alpha = (1, 0, 0), \beta = (1, 0, 0))$. The input values $x = (x_0, x_1, x_2)$ of the S-box correspond to the solution space of the following linear equations, which are derived from Equation 8.

$$\begin{cases} 1 = 1 \\ 0 = x_2 \oplus 1 \\ 0 = x_1 \end{cases}$$

## 3 Overview of Our Collision Attack Framework

Following the collision attack framework by Dinur *et al.*'s [8], our differential-based collision attacks consist of two essential steps: one step is to find a $n_1$-round high probability collision-generating differential trail (we call it the colliding trail for simplicity). The second step is to construct a $n_2$-round connector that promises a subspace of message pairs which satisfy the colliding trail at the same time and the constraints imposed by the padding rule and IV of Xoodyak hash mode as depicted in Figure 3.

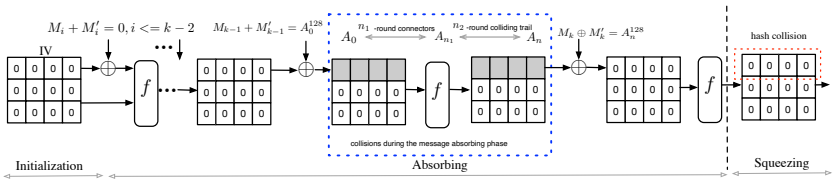


Fig. 3: Our Collision Attack Framework of Xoodyak Hash Mode

In the $n$-round differential trail search of sponge-based hash functions, the Constrained-Input Constrained-Output (CICO) problem [10] arises. The CICO

property imposes specific constraints on both the input and output differences. In our collision attack framework, there is no difference in the first $k-1$ message blocks, where the messages $M_0$, $M_1$, $\cdots$, $M_{k-2}$ are chosen randomly. We focus our analysis on collisions occurring during the absorbing phase of the hash mode. Thus, the last 256 bits of the input difference $A_0$ and the output difference $A_n$ are constrained to be zero.

In our attack framework depicted in Figure 3, assume there are $k+1$ message blocks, denoted by $M_i$ for $i \in \{0, 1, ..., k\}$. During the middle absorbing phase, we introduce a message pair $(M_{k-1}, M'_{k-1})$ that satisfies the first 128 bits of the input difference $A_0$, denoted by $A_0^{128}$.

From this starting point, we generate an $n$-round differential trail, which includes a $n_2$-round colliding trail that ensure $h$-bit collision and a $n_1$-round differential trail (we refer to it as the connecting trail for simplicity) for constructing a connector that generates a subspace of messages linking the initial conditions of sponge construction and the input difference of the subsequent colliding trail.

Once we construct such a $n$-round differential trail and find a message pair $(M_0, \ldots, M_{k-1})$ and $(M'_0, \ldots, M'_{k-1})$ that follows this trail where $A_0^{128} = M_{k-1} \oplus M'_{k-1}$, we introduce one more pair of message blocks $(M_k, M'_k)$ at the last absorbing phase that has the same difference in the first 128 bits, *i.e.*, $A_n^{128} = M_k \oplus M'_k$, the matching difference is then canceled out, resulting in zero difference during the squeezing phase, thus successfully converting it into a real collision.

### 3.1 Complexity Analysis

Suppose a $n$-round collision attack comprises a $n_1$-round connecting trail with weight $w_1$ and a $n_2$-round colliding trail with weight $w_2$, where $n_1 \leq 2$. The overall complexity includes both the complexity of the connector construction phase, denoted as $T_1$, and the complexity of the exhaustive search phase, denoted as $T_2$.

**Connector Construction Complexity.** The core idea of constructing $n_1$-round connector is to convert the problem to solving an algebraic system. For the attacker, the controllable initial degrees of freedom (DF) is 128.

Assume that an algebraic system of $n_1$-round connector is constructed, where the number of the linear equations denote by $n_l$, and the number of non-linear equations denote by $n_q$. The number of non-linear equations that can be linearized by guessing $d_f$ extra equations (we call them guess equations which consume $d_f$ DF) is denoted by $n'_l$. Thus, the final linear system includes $n_l + n'_l + d_f$ linear boolean equations in 128 Boolean variables.

Let us first recall the equivalent conversion of implementation efficiency between connector construction and $n$-round Xoodoo. The total number of bit operations[3] for an $n$-round Xoodoo is given by $n \times (1024 + 768 + 32) = 1824n$. If $n = 3$, The total number of bit operations $\approx 2^{12.4178}$.

[3] where $2\times128+2\times384 = 1024$ bit operations are for the $\theta$ operation, $2\times3\times128 = 768$ bit operations are for the $\chi$ operation, and 32 bit operations bit operations are for the $\iota$ operation.

Regarding this linear system, it is composed of $n_l + n'_l + d_f$ linear equations in 128 variables. Assume $n_l + n'_l + d_f \leq 128$, then the system has a non-trivial solution. The time complexity of Gaussian elimination $T_g$ for a system of linear equations is approximately

$$T_g \approx \frac{1}{3}(n_l + n'_l + d_f)^2 \times 128.$$

Consequently, the complexity of solving a linear system can be calculated by taking the ratio of the number of bit operations required for one iteration of Gaussian elimination to the total number of bit operations in $n$-round Xoodoo calls, *i.e.* $\frac{T_g}{1824n}$. During each iteration of Gaussian elimination, it is expected to obtain $2^{128-n_l-n'_l-d_f}$ solutions. The verification time complexity is estimated as $T_v = 2^{128-n_l-n'_l-d_f}$.

As each guess equation can be assigned a constant value of either 0 or 1. In this way, for one connector, we can construct $2^{d_f}$ linear systems.

To satisfy the remaining $n_q - n'_l$ non-linear equations and the $w_2$ conditions of the difference-value of the colliding trail, we need to prepare at least $2^{w_2+n_q-n'_l}$ solutions. Thus, the time complexity corresponds to the time required to construct $2^{(w_2+n_q-n'_l)-(128-n_l-n'_l-d_f)-d_f} = 2^{w_2+n_q+n_l-128}$ connectors.

The total time complexity in connector construction is equivalent to,

$$T_1 = 2^{w_2+n_q+n_l-128} \times (2^{d_f} \times (\frac{T_g}{1824n} + T_v)) \tag{9}$$

**Exhaustive Search Complexity.** The remaining $n_q - n'_l$ equations and the $w_2$ conditions on the difference values of the colliding trail are satisfied via exhaustive search. In the exhaustive search phase of the attack, we try $2^{w_2+n_q-n'_l}$ different message pairs in order to find a pair whose difference evolves according to the specified $n_2$-round colliding trail. The time complexity is

$$T_2 = 2^{w_2+n_q-n'_l} \tag{10}$$

In total, the time complexity denoted by $T_{total}$ of the $n$-round collision attack is

$$\begin{aligned} T_{total} &= T_1 + T_2 \\ &= 2^{w_2+n_q+n_l+d_f-128} \cdot (\frac{T_g}{1824n} + T_v) + 2^{w_2+n_q-n'_l} \end{aligned} \tag{11}$$

### 3.2 *n*-round Differential Trail Search

Differential cryptanalysis is one of the most effective methods to attack cryptographic permutations or block ciphers. It was first proposed by Biham and Shamir in 1991 [4]. The idea of differential cryptanalysis is to choose an input difference such that output differences can be reliably predicted. If a certain input/output difference exists with an optimal probability, it can be used to construct a distinguisher. This distinguisher can then be used to recover the initial state in a collision attack or the key in a key recovery attack.

For the study of the differential trail propagation of Xoodoo, we take a $n$-round differential trail as an example. A $n$-round differential trail starting from 0-th round has the following form:

$$A_0 \xrightarrow{\theta} B_0 \xrightarrow{\rho_{west}} C_0 \xrightarrow{\chi} D_0 \xrightarrow{\lambda} C_1 \xrightarrow{\chi} \ldots \xrightarrow{\lambda} C_{n-1} \xrightarrow{\chi} D_{n-1} \xrightarrow{\rho_{east}} A_n \tag{12}$$

With $i$ indexed from 0, let $C_i$ denote the input difference and $D_i$ denote the output difference of the $i$-th $\chi$ mapping. The last 256 bits of both the input and output differences (*i.e.*, $A_0, A_n$) are zero. We denote the linear layer as $\lambda = \rho_{\text{west}} \circ \theta \circ \rho_{\text{east}}$.

The weight of the $j$-th S-box in the $i$-th round is denoted by $w_i^j$, where $i$ starts from 0. The weight of a $n$-round differential trail is defined as the sum of the weights of all 3-bit S-boxes:

$$w(A_0 \to A_n) = \sum_{i=0}^{r-1} \sum_{j=0}^{127} w_i^j$$

For all valid difference patterns, we have $w_{(c,d)} = 2$. This can be represented as twice the total number of active S-boxes (non-zero difference) in either $C^k$ or $D^k$ for simplicity.

$$w(A_0 \to A_n) = 2 \times \sum_{i=0}^{r-1} \sum_{j=0}^{4} \sum_{k=0}^{32} s_i[j][*][k] \tag{13}$$

In this context, $s_i[j][*][k]$ indicates the active status of the $32j + k$-th S-box in the $i$-th round, taking a value of 1 if it is active and 0 otherwise.

### 3.3 Loop Differential Trail

In this section, we introduce a specific differential trail with internal differences that consist solely of *loop* elements, based on the differential properties of the round function of Xoodoo. We now provide a formal definition of such a differential trail as follows.

**Definition 4 (Loop Differential Trail).** *For a $n$-round differential trail, if all internal differences consist solely of loop elements, we refer to such a $n$-differential trail as a loop differential trail.*

It can be easily observed from Equation 2 and Equation 4 that, in order to eliminate the influence of $\rho_{west}$ and $\rho_{east}$, only the first plane of $S[x][0][z]$ is allowed to have a non-zero difference. To ensure that only the first plane of the entire state retains a non-zero difference after the non-linear layer $\chi$, we restrict the types of valid differential patterns. Specifically, we only allow $(C[x][0][z], C[x][1][z], C[x][2][z]) = (1, 0, 0)$ and $(D[x][0][z], D[x][1][z], D[x][2][z]) = (1, 0, 0)$.

For giving a more formal definition of the weight of $n$-round *loop* differential trail, we first define the notion of valid configuration of active S-boxes in Definition 5 (it is also suitable for general differential trails).

**Definition 5 (Valid Configuration of Active S-boxes).** *Let $r \geq 2$ be the number of rounds. We say a configuration $(s_0, \ldots, s_{r-1})$ is valid if there exists a $r$-round differential trail $(A_0, A_1, \ldots, A_r)$ with $s_0, \ldots, s_{r-1}$ active S-boxes at round 0, $\ldots$, $r-1$, respectively.*

With these constraints, we ensure that the output difference $A_0$ maintains the same differential state even after propagating through multiple rounds. For example, a one-round loop differential trail with a valid one-round configuration of active S-boxes ($s_0 = 32$) is illustrated in Figure 4. In this way, we can extend it to a 2-round loop differential trail with the configuration ($s_0 = 32, s_1 = 32$), and to a 3-round loop differential trail with the configuration ($s_0 = 32, s_1 = 32, s_2 = 32$). In what follows, we describe how to use these *loop* trails to find practical collision results.

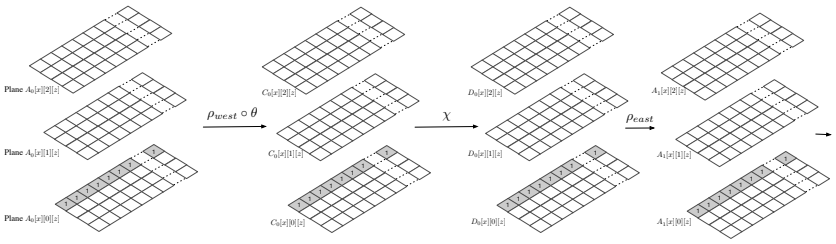


Fig. 4: 1-round Loop Differential Trail

## 4 SAT-based Collision Search

In this section, we develop a SAT-based collision search method. This method effectively reduces the practical collision search problem into a SAT problem.

### 4.1 Overview of Our Collision Search Method

To search for a valid differential trail and conforming message pair simultaneously, Liu *et al.* [14] at the CRYPTO 2020 developed a novel MILP model. Their MILP model independently constructed the difference transition and value transition of the target primitive as well as the difference-value relations model through the non-linear layer. Based on this MILP model, they present the first practical Semi-Free-Start (SFS) collision for 6-round Gimli-Hash.

Inspired by Liu *et al.*'s approach [14], we propose a SAT model for collision search. Our model construct the value transitions model and the difference-value relations model through the non-linear layer simultaneously. For the difference-value relations model, we utilize Property 3 to derive the implicit difference-value relations for each round of Xoodoo. It is noted that our model does not require a differential transition model; instead, the known differential trails are used as the constants.

We choose SAT tool for two main reasons. First, SAT is particularly well-suited for describing bit-level operations. At AISACRYPT 2022, Guo *et al.* [12] proved significant advantages of SAT models over MILP models when applied to differential trail search for KECCAK-$f$[1600]. In [7], Mathieu *et al.* reconstructed the MILP-based search model for the ASCON MitM attack proposed by Qin *et al.* [20] using a SAT model. This reconstruction not only significantly accelerated the solving process but also eliminated the need for heuristic algorithms, such as those relying on weak diffusion structures. To further study the impact of model and solver selection on cryptanalysis tasks, Bellini *et al.* [2] built SAT, SMT, MILP, and CP search models for various bit-oriented cryptographic primitives including block ciphers, permutations, and hash functions—targeting tasks such as optimal differential trail search, enumerating all optimal trails, and estimating differential probabilities. Their findings confirmed that SAT solvers generally outperform others for ARX primitives and SPN structures, demonstrating significant advantages.

On the other hand, the Sage [26] library includes a module named *sage.sat*, specifically designed for handling SAT problems. This module provides CNF encoding tools and functionalities for working with Boolean expressions and solving problems related to their satisfiability. Based on the *sage.sat* module, we can conveniently construct the corresponding SAT model for the linear layer as well as for the difference-value relations. Furthermore, it allows us to compatibly describe constraint models for nonlinear layers using truth table-based encoding methods [1,25]. Through this approach, we present an efficient SAT-based collision search model.

In the next section, we will provide a detailed explanation of the construction process for the SAT-based collision search model.

### 4.2 Constructing SAT Model for Value Transitions

The XOODOO permutation updates the initial state of 384 bits by applying the round function over a total of $n$ rounds. A $n$-round value transition model can be expressed in the following form:

$$a_0 \xrightarrow{\theta} b_0 \xrightarrow{\iota\circ\rho_{west}} c_0 \xrightarrow{\chi} d_0 \xrightarrow{\rho_{east}} a_1 \xrightarrow{\theta} b_1 \xrightarrow{\iota\circ\rho_{west}} c_1 \xrightarrow{\chi} d_1 \ldots \xrightarrow{\chi} d_{n-1} \tag{14}$$

**Modeling the Initialization.** In terms of the SAT model, each round requires four sets $a_i, b_i, c_i, d_i$ of 384 Boolean variables, with $b_i = \theta(a_i)$, $c_i = \iota \circ \rho_{west}(b_i)$, $d_i = \chi(c_i)$, $a_{i+1} = \rho_{east}(d_i)$. We omit the last $\rho_{east}$, as it does not influence our analysis. The initial state of XOODYAK hash mode in our model is denoted by $a_0$, the last 256 Boolean variables are assigned constant values according to the known padding rule and color bits.

**Modeling the Linear Mappings.** Linear mappings include $\theta$, $\rho_{\text{west}}$, $\rho_{\text{east}}$, and $\iota$, which involve a large number of XOR operations and cyclic shifts. We utilize the *sage.sat* module of SageMath [26] to simplify the cumbersome encoding of CNF clauses for the linear layer.

By simulating the linear mappings through symbolic computation, we generate the Boolean expressions of the internal state over $n$ rounds. Using the *sage.sat* module, we can convert these Boolean expressions into the corresponding CNF clauses.

This approach enhances the flexibility of directly characterizing the SAT model for linear mappings but also broadens our capabilities for CNF clauses encoding of multivariate XOR operations. The specific process for modeling the linear layer is outlined in Algorithm 1.

**Algorithm 1** Linear Layer SAT Model

**Require:** Input $a_i, b_i, c_i, d_i$, $0 \leq i < n$, linear mappings $\theta, \rho_{west}, \iota, \rho_{east}$, number of rounds $n$.
**Ensure:** DIMACS format of the CNF clauses for the $n$-round iteration.
1: Initialize $Q = \emptyset$;
2: **for** $i$ from 0 to $n$ **do**
3: Compute the Boolean expressions $a_i \leftarrow \theta(a_i)$;
4: Add $b_i \oplus a_i$ to the set $Q$;
5: Compute the Boolean expressions $b_i \leftarrow \iota \circ \rho_{west}(b_i)$;
6: Add $c_i \oplus b_i$ to the set $Q$;
7: **if** $i < n - 1$ **then**
8: Compute the Boolean expressions $d_i \leftarrow \rho_{east}(d_i)$;
9: Add $a_{i+1} \oplus d_i$ to the set $Q$;
10: **end if**
11: **end for**
12: Convert the set $Q$ into CNF clauses in DIMACS format using the *sage.sat* module.

**Modeling the Non-Linear Mapping $\chi$.** According to Property 5, the lookup table of 3-bit $\chi$ is given in Table 4, where $x = (x_0, x_1, x_2)$; if $x = 4$, then $(x_0 = 1, x_1 = 0, x_2 = 0)$.

Table 4: Xoodoo 's 3-bit S-box $\chi$ as a lookup table.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\chi(x)$ | 0 | 5 | 3 | 2 | 6 | 1 | 4 | 7 |

The general encoding method of $\chi$ is to extract all items of Table 4 into a truth table and feed this table into existing mathematical software to automatically generate minimized CNFs. For small-scale problems, such as those where the size of S-box is less than 5, we recommend using Logic Friday [4] which is a free tool specifically designed for Boolean logic analysis. Unfortunately, it only supports Windows systems. Fortunately, the POSform function[5] provided by the

[4] please refer to `http://sontrak.com/`.
[5] please refer to `https://docs.sympy.org/latest/modules/logic.html`.

**Algorithm 2** Difference-Value Relations SAT Model

**Require:** $c_i, 0 \leq i < n$, the non-linear operations $\chi^{n-1} \circ \cdots \circ \chi^0$, the number of rounds $n$, a given differential trail $(A_0, B_0, C_0, D_0, A_1, \ldots, A_n)$(please refer to trail model 12), and an auxiliary binary variable $u$.
**Ensure:** The value of $a_0$ or "Invalid".
1: Initialize a set $Q = \emptyset$;
2: **for** $i$ from 0 to $n$ **do**
3: $f^i = c_i \oplus uC_i$
4: Compute the output of the nonlinear operation: $f^{i+1} \leftarrow \chi^i(f^i)$;
5: Add $\mathcal{D}_u f^{i+1} \oplus D_i$ to the set $Q$;
6: **end for**
7: Convert the set $Q$ into CNF clauses in DIMACS format using the *sage.sat* module.);

logic module in the SymPy library can achieve the same results and the logic module of SymPy, both of which employ the Quine-McCluskey algorithm [21,16] for minimizing logical functions. For larger-scale problems, particularly when the size of the S-box exceeds 5, the Espresso tool can be utilized.

### 4.3 Constructing Difference-Value Relations Through $\chi$

The difference transitions and the value transitions are dependent only on the non-linear operation. For the $(j \times k)$-th S-box in the $i$-round, we denote the input difference and the output difference as $C_i[j][*][k]$ and$D_i[j][*][k]$, respectively, where $0 \leq i < n$, $0 \leq j < 4$, $0 \leq k < 32$.

Our goal is to derive all relations between the differential pattern ($C_i[j][*][k]$, $D_i[j][*][k]$) and the input value $c_i[j][*][k]$. Based on Property 3, the difference-value relations can be expressed as the following Boolean equations,

$$\begin{cases} D_i[j][0][k] \\ = \mathcal{D}_u\chi_0(c_i[j][0][k] \oplus uC_i[j][0][k], c_i[j][1][k] \oplus uC_i[j][1][k], c_i[j][2][k] \oplus uC_i[j][2][k]) \\ D_i[j][1][k] \\ = \mathcal{D}_u\chi_1(c_i[j][0][k] \oplus uC_i[j][0][k], c_i[j][1][k] \oplus uC_i[j][1][k], c_i[j][2][k] \oplus uC_i[j][2][k]) \\ D_i[j][2][k] \\ = \mathcal{D}_u\chi_2(c_i[j][0][k] \oplus uC_i[j][0][k], c_i[j][1][k] \oplus uC_i[j][1][k], c_i[j][2][k] \oplus uC_i[j][2][k]) \end{cases} \tag{15}$$

We utilize SageMath as the symbolic computation tool to construct difference-value Boolean equations and use the sage.sat module to convert Boolean equations to CNF clauses in DIMACS format. The specific process for modeling the difference-value relations is presented in Algorithm 2.

## 5 New Collision Attacks on Xoodyak Hash Mode

In this section, we present the first 2-round collision attack and a 3-round SFS collision attack based on our SAT-based collision search model.

### 5.1 2-round Collision Attack

Our 2-round collision search can be divided into the following two phases:

1. Prepare a 2-round *loop* differential trail $(A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1)$ with the configuration $(s_0 = 32, s_1 = 32)$. In each 384-bit difference state, only the first 32 bits have non-zero difference.
2. Construct a 2-round collision search SAT model and invoke SAT solver to find a conforming message pairs satisfying the 2-round loop differential trail. This message pair can be viewed as actual colliding pairs.

**Theoretical Complexity Analysis.** The weight of the second round trail is $32 \times 2 = 64$. Based on the 1-round connector, we can construct a linear system with $n_l = 64$ linear equations from the difference-value relations. The remaining DF is 64, such that 1-round connector generates a subspace of $2^{64}$ message pairs. This enables us to exhaustively enumerate the message pairs generated with the connectors until we find a message pair that following our 1-round colliding trail. The time complexity of 2-round collision attack $T_{total}$ is $2^{64}$. Actually, $T_{total}$ can be further reduced to $2^{46.6474}$ using the parameters $n_l = 64$, $n_q = 64$, $n'_l = 19$, and $d_f = 19$ based on our linearization strategy outlined in Section 5.3, by directly constructing the 2-round *loop* differential trails as 2-round connectors.

In the practical collision search on the 2-round Xoodyak hash mode, it typically requires at least $2^{46.6474}$ executions of the Xoodyak hash function to find an actual collision. In contrast, our SAT model demonstrates outstanding performance by enabling us to find a 2-round collision within minutes.

**Practical 2-round Collision Search.** We use different SAT solvers to solve our 2-round collision search model, the `Treengeling` solver performs well as shown in Appendix A (see Table 6).It takes only 64.7 seconds to find a practical collision. Thereal colliding pair is listed in Table 7.

### 5.2 Practical 3-round SFS Collision Search

For the 3-round loop differential trail with weight 192, finding a practical collision is challenging. In this section, we implement a practical SFS collision attack on the 3-round Xoodoo hash mode. The theoretical collision attack will be analyzed in detail in Section 5.3.

In the Semi-Free-Start (SFS) collision attack, the attacker has the ability to select the initial chain value, specifically the initialization vector (IV), along with a pair of distinct messages, denoted as $M_0 \| IV$ and $M'_0 \| IV$. The goal is to find these messages such that their hash values, calculated with the chosen IV, are equal: $H(IV, M_0) = H(IV, M'_0)$ [24].

The 3-round SFS collision search is conducted through the following steps:

1. Prepare a 3-round *loop* differential trail $(A_0, B_0, C_0, D_0, A_1, \ldots, D_2)$ with the configuration $(s_0 = 32, s_1 = 32, s_2 = 32)$. In each 384-bit difference state, only the first 32 bits have non-zero difference.

2. Construct a 3-round SFS collision search SAT model without imposing any constraints on the last 256 Boolean variables of the initial state $a_0$. We then invoke a SAT solver to find a conforming message pair that satisfy the 3-round loop differential trail. This message pair can be regarded as actual SFS colliding pair.

After 3-round SFS collision search model constructed, We use `Treengeling` [3] to solve it, it takes just 3.67s to find an actual 3-round SFS colliding pair. This actual colliding pair is presented in Table 5. However, for 4-round loop trail, we are unable to obtain an actual 4-round SFS colliding pair within a reasonable time (limited to 2 days).

Table 5: 3-round Practical SFS Collision for XOODYAK Hash Mode ($\bar{M}_0, \bar{M}'_0$)

| $\bar{M}_0$ |
|---|
| 0x89520d63 0x81dd4a10 0xfd153620 0x5f643139 |
| 0x87f72cf8 0x226e3ce7 0x6dd82fcb 0xc42eee67 |
| 0x683125fd 0xbd69af3d 0x859690da 0x84a2196b |
| $\bar{M}'_0 \oplus A_0$ |
| 0x76adf29c 0x81dd4a10 0xfd153620 0x5f643139 |
| 0x87f72cf8 0x226e3ce7 0x6dd82fcb 0xc42eee67 |
| 0x683125fd 0xbd69af3d 0x859690da 0x84a2196b |

### 5.3 Improved 3-round Collision Attacks

Now, we have a 3-round *loop* differential trail with the configuration of ($s_0 = 32, s_1 = 32, s_2 = 32$). For the last two rounds, the total weight equals the birthday bound (128 for both XOODOO-HASH and XOODOO-XOF). It is impractical to randomly select a 2-round colliding trail and construct a 1-round connector.

To make the collision attack on the XOODYAK hash mode theoretically feasible for up to 3 rounds, our 3-round collision attack is constructed using a 2-round connector and a 1-round colliding trail with weight 64. We propose a linearization strategy in this section to partially linearize 64 quadratic conditions of the second-round connectors. Two rounds of XOODOO permutation are expressed as

$$a_0 \xrightarrow{\iota\circ\rho_{west}\circ\theta} c_0 \xrightarrow{\chi} d_0 \xrightarrow{\iota\circ\rho_{west}\circ\theta\circ\rho_{east}} c_1 \xrightarrow{\chi} d_1 \xrightarrow{\rho_{east}} a_1 \tag{16}$$

In our 3-round *loop* trail, all S-boxes have the same valid difference pattern, $(1, 0, 0, 1, 0, 0)$, in the first round, we have $n_l = 64$ linear equations derived from the difference-value relations $D_0 = \mathcal{D}_u\chi(c_0 \oplus uC_0)$, *i.e.*, $c_0[0][1][z] = 0$ and $c_0[0][2][z] = 1$, where $0 \le z < 32$. In the second round, the difference-value relations $D_1 = \mathcal{D}_u\chi(c_1 \oplus uC_1)$ give rise to $n_q = 64$ quadratic equations, *i.e.*, $c_1[0][1][z] = 0$ and $c_1[0][2][z] = 1$. Hence, the total complexity of 3-round collision attack can be computed based on Equation 11,

$$T_{total} = 2^{51.5822+d_f} \times \frac{(64 + n'_l + d_f)^2 \times 128}{3} + 2^{128-n'_l} + 2^{128-n'_l} \tag{17}$$

**Linearization Strategy** Our aim is to maximize the number of quadratic equations $n'_l$ that can be linearized while minimizing the consumption of degrees of freedom is essential. This will reduce the overall complexity of collision attack.

As the inputs are already grouped to subset with the 64 linear constraints of differentials, the first 32 active S-boxes are already fully linearized. Thus, the output bits $d_0[0][*][z]$ of the 32 active S-boxes $c_0[0][*][z]$ after the first $\chi$ mapping must be linear bits *i.e.*, $(d_0[0][0][z], d_0[0][1][z], d_0[0][2][z], 0 \leq z < 32)$.

Let us consider a given state $a_0$, where only the first 128 bits are treated as Boolean variables, while the remaining 256 bits are fixed as constants. After propagation through the linear layer $\iota \circ \rho_{\text{west}} \circ \theta$, the resulting state $c_0$ satisfies the following property:

*Property 4.* If the bit $c_0[0][1][z]$ is imposed a linear equation, then the bit $c_0[3][2][z]$ is also imposed a linear equation equivalent to bit $c_0[0][1][z+11]$, where $0 \leq z < 32$.

*Property 5.* If the bit $c_0[0][2][z]$ is imposed a linear equation, then the bit $c_0[3][2][z]$ is also imposed a linear equation equivalent to bit $c_0[0][2][z-11]$, where $0 \leq z < 32$.

This relationship is determined by the combined effects of the linear operations $\theta$-effect and $\rho_{west}$. The detailed proof is omitted here for brevity. A graphical explanation for the linearization can be seen from Figure 5.

Based on Property 2, any two bit of the output can be linearized by adding one extra linear equations on the input. Thus, extra $2 \times 2 \times 32 = 128$ bits in $d_0$ can be linearized. We introduce 384 Boolean variables $x_{384}, \ldots, x_{767}$ to represent each bit of $d_0$. Among these, $96 + 128 = 224$ bits' ANF expressions are linear expressions with respect to the inputs $x_i$,$0 \leq i < 128$. We refer to such bits as linear bits. In contrast, the remaining 160 bits are referred to as non-linear bits, as their ANF expressions are non-linear with respect to the inputs $x_i$, $0 \leq i < 128$.

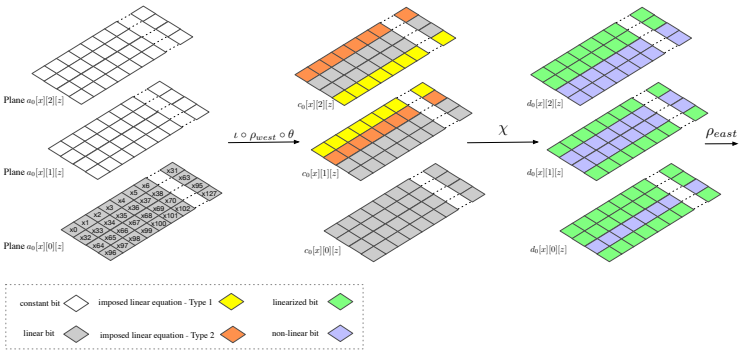


Fig. 5: Propagation of Boolean Variables in the First Round

Specially, for non-linear bits $x_i$, $704 \leq i \leq 735$, each non-linear bit require one extra linear equation to achieve linearization, resulting in the consumption of 1 DF. We list the quadratic terms related to the non-linear bits below.

$$\begin{cases} x_{i+686} : (x_i + x_{i+9})(x_{i+32} + x_{i+41} + x_{i+46}), 18 \leq i \leq 22 \\ x_{i+709} : (x_i + x_{i+23})(x_{i+32} + x_{i+55} + x_{i+69}), 0 \leq i \leq 8 \\ x_{i+718} : (x_i + x_{i+9})(x_{i+32} + x_{i+41} + x_{i+78}), 0 \leq i \leq 17 \end{cases}$$

For example, if we set $x_7 + x_{30} = c$, where $c = 0$ or $c = 1$, $x_{716}$ can be linearized. In this way, this process enables us to obtain one linearized quadratic equation at the cost of 1 DF, which can be mathematically expressed as $d_f = n'_l$.

Based on the above analysis, we can re-write Equation 17 as below,

$$T_{total} = 2^{51.5822+n'_l} \times \frac{(64 + 2n'_l)^2 \times 128}{3} + 2^{128-n'_l+1} \tag{18}$$

The above computation has the optimal choice when $n'_l = 29$, the time complexity of the 3-round collision attack reaches its minimum value, *i.e.*, $T_{total} = 2^{100.9311}$.

**Improved 2-round Theoretical Complexity Analysis.** The time complexity of 2-round collision attack $T_{total}$ can be further reduced to $2^{46.6474}$ by directly constructing the 2-round *loop* differential trails as connectors for the two rounds. With the same linearization strategy, we set $n_l = 64, n_q = 64$ according to the weight of 2-round *loop* differential trail, so that $T_{total}$ reaches its minimum value when $n'_l = 19, d_f = 19$, specially $2^{46.6474}$.

## 6 Conclusion

In this paper, we observe several key properties of the Xoodoo round function, particularly its unique $\theta$ and non-linear operation $\chi$. Leveraging these properties, we design special differential trails referred to as *loop* differential trails to improve differential-based collision attacks. To quickly find practical collisions, we build a SAT-based collision search model for the Xoodyak hash mode. It would be interesting to apply this model to other permutation-based cryptographic primitives. We present an actual 2-round collision and a 3-round SFS collision on the Xoodyak hash mode. Furthermore, by employing a linearization strategy, we reduce the best-known time complexity for 3-round collision attacks to $2^{100.9311}$. To the best of our knowledge, these are the best attacks on the round-reduced Xoodyak hash mode, covering both Xoodyak-`hash` and Xoodyak-Xof.

## Acknowledgments

The authors would like to thank Kai Hu for all valuable feedback. We are grateful to the anonymous reviewers for their valuable comments. This work was supported by the National Key Research and Development Program of China under Grant 2023YFB3106501.

## References


1. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) S-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017)
2. Bellini, E., Piccoli, A.D., Formenti, M., Gérault, D., Huynh, P., Pelizzola, S., Polese, S., Visconti, A.: Differential cryptanalysis with SAT, SMT, MILP, and CP: a detailed comparison for bit-oriented primitives. In: CANS. Lecture Notes in Computer Science, vol. 14342, pp. 268–292 (2023)
3. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In: Balyo, T., Heule, M., Järvisalo, M. (eds.) Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2017-1, pp. 14–15 (2017)
4. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of CRYPTOLOGY **4**(1), 3–72 (1991)
5. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xoofff. IACR Transactions on Symmetric Cryptology **2018**, 1–38 (2018)
6. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. IACR Transactions on Symmetric Cryptology pp. 60–87 (2020)
7. Degré, M., Derbez, P., Lahaye, L., Schrottenloher, A.: New models for the cryptanalysis of Ascon. IACR Cryptol. ePrint Arch. p. 298 (2024)
8. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on Keccak-224 and Keccak-256. In: Fast Software Encryption - 19th International Workshop, FSE 2012. vol. 7549, pp. 442–461. Springer, Berlin, Heidelberg (2012)
9. Dong, X., Zhao, B., Qin, L., Hou, Q., Zhang, S., Wang, X.: Generic MitM attack frameworks on sponge constructions. In: CRYPTO (4). Lecture Notes in Computer Science, vol. 14923, pp. 3–37. Springer (2024)
10. Guido, B., Joan, D., Michaël, P., Gilles, V.: Cryptographic sponge functions. `https://keccak.team/files/CSF-0.1.pdf` (2011)
11. Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L.: Practical collision attacks against round-reduced SHA-3. Journal of Cryptology **33**(1), 228–270 (2020)
12. Guo, J., Liu, G., Song, L., Tu, Y.: Exploring SAT for cryptanalysis:(quantum) collision attacks against 6-round SHA-3. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 645–674. Springer (2022)
13. Huang, S., Ben-Yehuda, O.A., Dunkelman, O., Maximov, A.: Finding collisions against 4-round SHA3-384 in practical time. IACR Cryptol. ePrint Arch. p. 194 (2022)
14. Liu, F., Isobe, T., Meier, W.: Automatic verification of differential characteristics: application to reduced Gimli. In: Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference. vol. 12172, pp. 219–248. Springer, Cham (2020)
15. Liu, M., Lu, X., Lin, D.: Differential-linear cryptanalysis from an algebraic perspective. In: Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference. vol. 12827, pp. 247–277. Springer, Cham (2021)
16. McCluskey, E.J.: Minimization of boolean functions. The Bell System Technical Journal **35**(6), 1417–1444 (1956)
17. Mella, S., Daemen, J., Assche, G.V.: Tighter trail bounds for Xoodoo. IACR Trans. Symmetric Cryptol. **2023**(4), 187–214 (2023)

18. NIST: The NIST lightweight cryptography project (2018), https://csrc.nist.gov/Projects/lightweight-cryptography
19. Qiao, K., Song, L., Liu, M., Guo, J.: New collision attacks on round-reduced Keccak. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 10212, pp. 216–243 (2017)
20. Qin, L., Hua, J., Dong, X., Yan, H., Wang, X.: Meet-in-the-middle preimage attacks on sponge-based hashing. In: EUROCRYPT (4). Lecture Notes in Computer Science, vol. 14007, pp. 158–188 (2023)
21. Quine, W.V.: A way to simplify truth functions. The American mathematical monthly **62**(9), 627–631 (1955)
22. Song, L., Guo, J.: Cube-attack-like cryptanalysis of round-reduced Keccak using MILP. IACR Trans. Symmetric Cryptol. **2018**(3), 182–214 (2018)
23. Song, L., Liao, G., Guo, J.: Non-full Sbox linearization: Applications to collision attacks on round-reduced Keccak. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 10402, pp. 428–451. Springer (2017)
24. Stevens, M., Karpman, P., Peyrin, T.: Freestart collision for full SHA-1. In: Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques. vol. 9665, pp. 459–483. Springer, Berlin, Heidelberg (2016)
25. Sun, L., Wang, W., Wang, M.: More accurate differential properties of LED64 and Midori64. IACR Trans. Symmetric Cryptol. **2018**(3), 93–123 (2018). https://doi.org/10.13154/tosc.v2018.i3.93-123
26. The Sage Developers: SageMath, the sage mathematics software system (version 9.5s) (2022), https://www.sagemath.org
27. Zhou, H., Li, Z., Dong, X., Jia, K., Meier, W.: Practical key-recovery attacks on round-reduced Ketje Jr, Xoodoo-AE and Xoodyak. Comput. J. **63**(8), 1231–1246 (2020)

# Appendix

## A Practical Collision Results

The real 2-round collision pair and the corresponding search time are listed as follows:

Table 6: 2-round Practical Collision Search Time

| practical collision search time | solver | thread |
|---|---|---|
| 64.7s | `Treengeling` | 10 |
| 966.4s | `Plingeling` | 10 |
| 1583.1s | `CryptoMiniSat` | 10 |
| 18983.3s | `CaDiCaL` | 1 |

Table 7: 2-round Practical Collision for Xoodyak Hash Mode ($\bar{M}_0, \bar{M}'_0$)

| $\bar{M}_0$ | | | |
|---|---|---|---|
| 0x2854a4be | 0x3adea611 | 0xffffffff | 0xaaaaaaab |
| 0x01000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00000000 | 0x00000000 | 0x00000000 | 0x00000001 |
| $\bar{M}'_0 \oplus A_0$ | | | |
| 0xd7ab5b41 | 0x3adea611 | 0xffffffff | 0xaaaaaaab |
| 0x01000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00000000 | 0x00000000 | 0x00000000 | 0x00000001 |

## B Quadratic Equations in the Second Round

To clearly illustrate the Boolean variables associated with the quadratic conditions, we present all quadratic equations in Table 8. We observe that each quadratic equation contains at least one non-linear bit or, in some instances, up to four non-linear bits.

Table 8: Statistics of 64 Quadratic Equations. $x(1,2,3) \triangleq x_1 \oplus x_2 \oplus x_3$; Orange bit is non-liner variable in $d_0$; Black bit is linear variable in $d_0$.

| No. | Quadratic Equations | No. | Quadratic Equations |
|---|---|---|---|
| 1 | $x(466, 475, 593, 602, 639, 650, 659)$ | 33 | $x(487, 496, 614, 623, 680, 703, 717)$ |
| 2 | $x(467, 476, 594, 603, 608, 651, 660)$ | 34 | $x(488, 497, 615, 624, 672, 681, 718)$ |
| 3 | $x(468, 477, 595, 604, 609, 652, 661)$ | 35 | $x(489, 498, 616, 625, 673, 682, 719)$ |
| 4 | $x(469, 478, 596, 605, 610, 653, 662)$ | 36 | $x(490, 499, 617, 626, 674, 683, 720)$ |
| 5 | $x(470, 479, 597, 606, 611, 654, 663)$ | 37 | $x(491, 500, 618, 627, 675, 684, 721)$ |
| 6 | $x(448, 471, 598, 607, 612, 655, 664)$ | 38 | $x(492, 501, 619, 628, 676, 685, 722)$ |
| 7 | $x(449, 472, 576, 599, 613, 656, 665)$ | 39 | $x(493, 502, 620, 629, 677, 686, 723)$ |
| 8 | $x(450, 473, 577, 600, 614, 657, 666)$ | 40 | $x(494, 503, 621, 630, 678, 687, 724)$ |
| 9 | $x(451, 474, 578, 601, 615, 658, 667)$ | 41 | $x(495, 504, 622, 631, 679, 688, 725)$ |
| 10 | $x(452, 475, 579, 602, 616, 659, 668)$ | 42 | $x(496, 505, 623, 632, 680, 689, 726)$ |
| 11 | $x(453, 476, 580, 603, 617, 660, 669)$ | 43 | $x(497, 506, 624, 633, 681, 690, 727)$ |
| 12 | $x(454, 477, 581, 604, 618, 661, 670)$ | 44 | $x(498, 507, 625, 634, 682, 691, 728)$ |
| 13 | $x(455, 478, 582, 605, 619, 662, 671)$ | 45 | $x(499, 508, 626, 635, 683, 692, 729)$ |
| 14 | $x(456, 479, 583, 606, 620, 640, 663)$ | 46 | $x(500, 509, 627, 636, 684, 693, 730)$ |
| 15 | $x(448, 457, 584, 607, 621, 641, 664)$ | 47 | $x(501, 510, 628, 637, 685, 694, 731)$ |
| 16 | $x(449, 458, 576, 585, 622, 642, 665)$ | 48 | $x(502, 511, 629, 638, 686, 695, 732)$ |
| 17 | $x(450, 459, 577, 586, 623, 643, 666)$ | 49 | $x(480, 503, 630, 639, 687, 696, 733)$ |
| 18 | $x(451, 460, 578, 587, 624, 644, 667)$ | 50 | $x(481, 504, 608, 631, 688, 697, 734)$ |
| 19 | $x(452, 461, 579, 588, 625, 645, 668)$ | 51 | $x(482, 505, 609, 632, 689, 698, 735)$ |
| 20 | $x(453, 462, 580, 589, 626, 646, 669)$ | 52 | $x(483, 506, 610, 633, 690, 699, 704)$ |
| 21 | $x(454, 463, 581, 590, 627, 647, 670)$ | 53 | $x(484, 507, 611, 634, 691, 700, 705)$ |
| 22 | $x(455, 464, 582, 591, 628, 648, 671)$ | 54 | $x(485, 508, 612, 635, 692, 701, 706)$ |
| 23 | $x(456, 465, 583, 592, 629, 640, 649)$ | 55 | $x(486, 509, 613, 636, 693, 702, 707)$ |
| 24 | $x(457, 466, 584, 593, 630, 641, 650)$ | 56 | $x(487, 510, 614, 637, 694, 703, 708)$ |
| 25 | $x(458, 467, 585, 594, 631, 642, 651$ | 57 | $x(488, 511, 615, 638, 672, 695, 709)$ |
| 26 | $x(459, 468, 586, 595, 632, 643, 652)$ | 58 | $x(480, 489, 616, 639, 673, 696, 710)$ |
| 27 | $x(460, 469, 587, 596, 633, 644, 653)$ | 59 | $x(481, 490, 608, 617, 674, 697, 711)$ |
| 28 | $x(461, 470, 588, 597, 634, 645, 654)$ | 60 | $x(482, 491, 609, 618, 675, 698, 712)$ |
| 29 | $x(462, 471, 589, 598, 635, 646, 655)$ | 61 | $x(483, 492, 610, 619, 676, 699, 713)$ |
| 30 | $x(463, 472, 590, 599, 636, 647, 656)$ | 62 | $x(484, 493, 611, 620, 677, 700, 714)$ |
| 31 | $x(464, 473, 591, 600, 637, 648, 657)$ | 63 | $x(485, 494, 612, 621, 678, 701, 715)$ |
| 32 | $x(465, 474, 592, 601, 638, 649, 658)$ | 64 | $x(486, 495, 613, 622, 679, 702, 716)$ |