IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 2, pp. 1–0. DOI:10.46586/tosc.v2022.i2.1-0

Integral Cryptanalysis of WARP based on Monomial Prediction

Hosein Hadipour and Maria Eichlseder

Graz University of Technology, Graz, Austria
hossein.hadipour@iaik.tugraz.at, maria.eichlseder@iaik.tugraz.at

Abstract. WARP is a 128-bit block cipher published by Banik et al. at SAC 2020 as a lightweight alternative to AES. It is based on a generalized Feistel network and achieves the smallest area footprint among 128-bit block ciphers in many settings. Previous analysis results include integral key-recovery attacks on 21 out of 41 rounds. In this paper, we propose integral key-recovery attacks on up to 32 rounds by improving both the integral distinguisher and the key-recovery approach substantially. For the distinguisher, we show how to model the monomial prediction technique proposed by Hu et al. at ASIACRYPT 2020 as a SAT problem and thus create a bit-oriented model of WARP taking the key schedule into account. Together with two additional observations on the properties of WARP's construction, we extend the best previous distinguisher by 2 rounds (as a classical integral distinguisher) or 4 rounds (for a generalized integral distinguisher). For the key recovery, we create a graph-based model of the round function and demonstrate how to manipulate the graph to obtain a cipher representation amenable to FFT-based key recovery.

Keywords: Lightweight cryptography · WARP · GFN · Integral cryptanalysis · Monomial prediction · CP · SAT · FFT key recovery

1 Introduction

Lightweight cryptographic primitives protect an increasing number of interconnected, highly resource-constrained devices transmitting sensitive information in our daily life, such as healthcare devices, the Internet of Things, or sensor networks. Cryptographers have designed a variety of new symmetric-key primitives fitting the constraints of these settings. The most prominent effort in this direction is the ongoing NIST LWC project aiming to standardize a lightweight authenticated encryption algorithm, as well as a lightweight hash function. Among the symmetric primitives, lightweight block ciphers have attracted attention as a building block of lightweight authenticated encryption schemes occupying a very small hardware footprint. While the first generation of lightweight block ciphers such as PRESENT [BKL+07] and LED [GPPR11] mainly focused on hardware footprint, more recent (tweakable) block cipher designs often target different criteria, such as low latency in QARMA [Ava17] or low energy consumption in MIDORI [BBI+15].

WARP is a lightweight 128-bit block cipher presented by Banik et al. at SAC 2020 [BBI+20] as a low-area drop-in replacement of AES-128. Such a lightweight, low-area alternative to AES-128 with the same interface size is attractive as it is easy to integrate in existing systems. Previous 128-bit designs like MIDORI and GIFT-128 [BPP+17] predominantly follow a Substitution-Permutation Network (SPN) structure. However, SPN ciphers are not perfect in terms of the area, particularly where a unified encryption and decryption circuit is required since designing fully involutory components with minimal area is challenging [GPV19]. Banik et al. thus designed WARP using the Generalized Feistel Structure (GFS) design paradigm which is involutory in nature. With its nibble-oriented

4-bit branches, the F-function consists only of a small 4-bit S-box followed by the key addition – an order of operations inspired by PICCOLO [SIH+11]. The resulting design shares similarities with LBlock [WZ11] and LBlock-like functions, but provides better diffusion of active S-boxes. At the same time, the design is significantly smaller than the prior ones for both encryption-only and unified encryption and decryption implementations.

The designers of WARP also provide security analysis against differential, linear, impossible differential, and integral attacks [BBI+20]. Regarding impossible differential attacks, they show a 21-round impossible differential distinguisher. Applying the method of Sasaki and Todo [ST17], we find that zero-correlation distinguishers also reach 21 rounds ($\boldsymbol{e}_{i+120} \nrightarrow \boldsymbol{e}_{i+60}$ and $\boldsymbol{e}_{i+56} \nrightarrow \boldsymbol{e}_{i+124}$ for $0 \leq i \leq 3$). For integral attacks, the designers argue that (nibble-oriented) integral distinguishers can cover no more than 20 rounds, and key-recovery attacks can extend these by at most 1 round. Additionally, there are a few third-party analysis results focusing primarily on WARP's differential properties [KY21a, TB21]. Teh and Biryukov [TB21] provided a more accurate analysis against differential attacks taking the clustering effect into account. They also investigate the security of WARP against boomerang attacks. The results are summarized in Table 1.

Table 1: Distinguishers and key-recovery attacks on round-reduced WARP. Gen. integral refers to generalized integral properties where the sum is taken over a function of the ciphertext bits rather than directly over the bits.

#R	Data	Time	Memory	Attack	Dist. / Key rec.	Reference
20	2^{124}	-	-	Integral	■ / □	[BBI+20]
20	$\mathbf{2^{123}}$	-	-	Integral	■ / □	Subsection 3.2
21	$\mathbf{2^{124}}$	-	-	Integral	■ / □	Subsection 3.2
22	2^{127}	-	-	Integral	■ / □	Subsection 3.2
22	$\mathbf{2^{123}}$	-	-	Gen. integral	■ / □	Subsection 4.1
23	$\mathbf{2^{124}}$	-	-	Gen. integral	■ / □	Subsection 4.1
24	2^{127}	-	-	Gen. integral	■ / □	Subsection 4.1
21	2^{124}	-	-	Integral	□ / ■	[BBI+20]
32	2^{127}	2^{127}	2^{108}	Integral	□ / ■	Section 4
18	$2^{104.62}$	-	-	Differential	■ / □	[TB21]
21	-	-	-	Impossible diff.	■ / □	[BBI+20]
21	-	-	-	Zero-correlation	■ / □	Section 1
21	2^{113}	2^{113}	2^{72}	Differential	□ / ■	[KY21b]
23	$2^{106.62}$	$2^{106.62}$	$2^{106.62}$	Differential	□ / ■	[TB21]
24	$2^{126.06}$	$2^{125.18}$	$2^{127.06}$	Rectangle	□ / ■	[TB21]

Our contributions. In this paper, we show how to improve both integral distinguishers and key-recovery techniques for WARP to extend the previous integral attack by 11 rounds to 32 (out of 41) rounds. While the previous distinguishers are based on a nibble-wise model, we provide a bit-oriented analysis using the recently introduced monomial prediction technique [HSWW20]. To apply the technique, we show how to encode the properties as a Constraint Programming (CP) or a SAT problem based on a Monomial Prediction Table (MPT). Our implementation is available in the following Github repository and can be applied to other designs as well:

https://github.com/hadipourh/mpt

Additionally, we offer two observations following from WARP's choice of adding the key after the S-box. Overall, this allows us to extend the previous integral distinguishers by 4 rounds. To the best of our knowledge, the resulting 24-round generalized integral distinguisher is the longest distinguisher for WARP.

We then extend the distinguisher to a 32-round key-recovery attack. Due to the high data complexity of 2^{127} queries for the distinguisher, we are quite constrained in the key-recovery phase, and need to carefully restructure the dependency graph in such a way that efficient key-recovery techniques such as the Meet-in-the-Middle (MitM) [SW12] and Fast Fourier Transform (FFT) [TA14] techniques become applicable. With a semi-automated graph-based approach, we still manage to add 8 rounds. The final attack complexity is dominated by the 2^{127} chosen-plaintext queries and thus close to generic attacks. If desired, the attack can easily be converted to a 31-round attack with complexity 2^{125}. In any case, the attack covers substantially more rounds than other published key-recovery attacks.

Outline. We recall the specification of WARP and background on integral cryptanalysis and monomial trails in Section 2. In Section 3, we show how to model monomial prediction for block ciphers as a CP or SAT problem, and apply our tool to find longer integral distinguishers for WARP. In Section 4, we extend our distinguisher to a key-recovery attack on 32-round WARP by representing many rounds as a directed acyclic graph to make them amenable to the MitM FFT key-recovery technique.

2 Preliminaries and Notation

In this section, we provide a brief specification of WARP and also introduce the notation used in this paper, as well as background on integral attacks and monomial prediction.

2.1 Specification of WARP

WARP is a 128-bit block cipher aiming at small-footprint circuit in the field of 128-bit lightweight block ciphers and follows a variant of the 32-branch GFS design paradigm. It receives a 128-bit plaintext with a 128-bit key and performs 40 full rounds plus 1 partial round to produce a 128-bit ciphertext. The internal state of WARP can be represented as $X = X_0 || \cdots || X_{31}$, where $X_i \in \{0,1\}^4$. WARP splits the 128-bit master key K into two 64-bit halves, i.e., $K = K^0 || K^1$, and $K^{(r-1) \mod 2}$ is used as the round-key in the rth round. The ith nibble of the round-key $K^{(b)}$ in round $b = (r-1) \bmod 2$ is denoted by $K_i^{(b)}$, where $b \in \{0,1\}$, and $0 \leq i \leq 15$. We denote the ith nibble in the input of rth round by $X_i^{(r-1)}$, where $1 \leq r \leq 41$ and $0 \leq i \leq 31$. We also sometimes use bitwise indexing, denoting the ith bit of $X^{(r-1)}$ counting from the left (MSB) by $x_i^{(r-1)}$, where $0 \leq i \leq 127$. When it is clear from the context, we only use a number between 0 and 127 to represent a certain bit of the internal state.

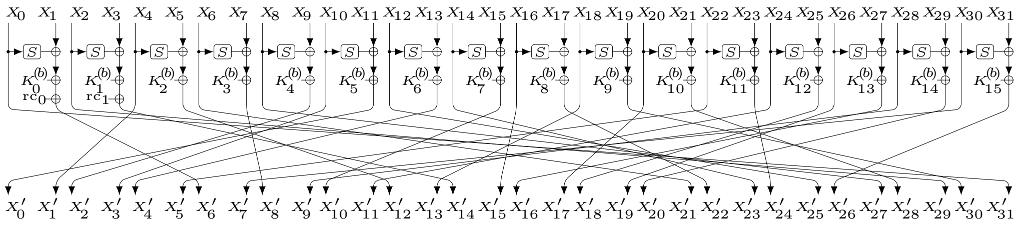
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10} X_{11} X_{12} X_{13} X_{14} X_{15} X_{16} X_{17} X_{18} X_{19} X_{20} X_{21} X_{22} X_{23} X_{24} X_{25} X_{26} X_{27} X_{28} X_{29} X_{30} X_{31}
S
$K_0^{(b)}$
$K_1^{(b)}$
$K_2^{(b)}$
$K_3^{(b)}$
$K_4^{(b)}$
$K_5^{(b)}$
$K_6^{(b)}$
$K_7^{(b)}$
$K_8^{(b)}$
$K_9^{(b)}$
$K_{10}^{(b)}$
$K_{11}^{(b)}$
$K_{12}^{(b)}$
$K_{13}^{(b)}$
$K_{14}^{(b)}$
$K_{15}^{(b)}$
rc_0
rc_1
X'_0 X'_1 X'_2 X'_3 X'_4 X'_5 X'_6 X'_7 X'_8 X'_9 X'_{10} X'_{11} X'_{12} X'_{13} X'_{14} X'_{15} X'_{16} X'_{17} X'_{18} X'_{19} X'_{20} X'_{21} X'_{22} X'_{23} X'_{24} X'_{25} X'_{26} X'_{27} X'_{28} X'_{29} X'_{30} X'_{31}

Figure 1: The round function of WARP.

As Figure 1 illustrates, the round function of WARP first applies the same S-box $S : \{0,1\}^4 \to \{0,1\}^4$ as well as the round-key addition on each of two consecutive nibbles of internal state. Next, a round constant is added and a permutation $\pi : \{0,\ldots,31\} \to \{0,\ldots,31\}$ is applied on the position of nibbles. The last round of WARP does not include the nibble permutation. Table 2 describes the nibble permutation π. Table 3 shows the lookup table of `WARP`'s S-box S, while Table 4 lists its Algebraic Normal Form (ANF). We refer to the `WARP` specification [BBI+20] for full details.

Table 2: Nibble permutation π of `WARP`.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	31	6	29	14	1	12	21	8	27	2	3	0	25	4	23	10
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(x)$	15	22	13	30	17	28	5	24	11	18	19	16	9	20	7	26

Table 3: 4-bit S-box S of `WARP`.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

Table 4: Algebraic Normal Form (ANF) of the `WARP` S-box S.

Bit 0, MSB	$x_0 x_1 x_3 + x_1 x_2 x_3 + x_0 x_1 + x_1 x_3 + x_2 x_3 + 1$
Bit 1	$x_0 x_1 x_2 + x_0 x_1 x_3 + x_1 x_2 x_3 + x_0 x_3 + x_0 + x_3 + 1$
Bit 2	$x_0 x_2 + x_0 x_3 + x_2 x_3 + x_0 + x_2$
Bit 3, LSB	$x_0 x_1 x_2 + x_0 x_1 x_3 + x_1 x_2 x_3 + x_0 x_2 + x_0 x_3 + x_1$

2.2 Boolean Functions

To represent bit vectors, we use bold italic lowercase letters, e.g., $\boldsymbol{x} \in \mathbb{F}_2^n$ denotes the n-bit vector $\boldsymbol{x} = (x_0, \cdots, x_{n-1})$. We also denote the n-bit zero vector by $\mathbf{0}$, and $\boldsymbol{e}_i$ represents a unit vector in which all coordinates are zero except for the ith coordinate which is equal to one. We also define a partial order over the set of n-bit vectors, such that for any n-bit vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{x} \leq \boldsymbol{y}$ if $x_i \leq y_i$ for all i. A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely represented by a multivariate polynomial in $\frac{\mathbb{F}_2[x_0,\ldots,x_{n-1}]}{\langle x_0^2+x_0,\ldots,x_{n-1}^2+x_{n-1}\rangle}$ which is called its *algebraic normal form* (ANF) and is defined as follows:

$$f(\boldsymbol{x}) = f(x_0, \ldots, x_{n-1}) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \cdot \left(\prod_{i=0}^{n-1} x_i^{u_i} \right), \quad \text{where} \quad a_{\boldsymbol{u}} \in \mathbb{F}_2.$$

To compactly represent a monomial $\prod_{i=0}^{n-1} x_i^{u_i}$, we use $\boldsymbol{x}^{\boldsymbol{u}}$ or $\pi_{\boldsymbol{u}}(\boldsymbol{x})$. Moreover, given a Boolean function $f = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \cdot \boldsymbol{x}^{\boldsymbol{u}}$, the coefficients of its ANF can be uniquely determined from its values and vice versa:

$$a_{\boldsymbol{u}} = \sum_{\boldsymbol{x} \leq \boldsymbol{u}} f(\boldsymbol{x}), \text{ and } f(\boldsymbol{u}) = \sum_{\boldsymbol{x} \leq \boldsymbol{u}} a_{\boldsymbol{x}}, \text{ for all } \boldsymbol{u} \in \mathbb{F}_2^n.$$

A vectorial Boolean function is a function $\boldsymbol{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ whose m coordinates are Boolean functions in n variables and it is compactly denoted by $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$. To show the presence and the absence of monomial $\pi_{\boldsymbol{u}}(\boldsymbol{x})$ in the ANF of $\pi_{\boldsymbol{v}}(\boldsymbol{y})$, we use $\pi_{\boldsymbol{u}}(\boldsymbol{x}) \to \pi_{\boldsymbol{v}}(\boldsymbol{y})$, and $\pi_{\boldsymbol{u}}(\boldsymbol{x}) \nrightarrow \pi_{\boldsymbol{v}}(\boldsymbol{y})$, respectively.

2.3 Integral Cryptanalysis

The idea of integral analysis was first introduced as a theoretical generalization of differential cryptanalysis by Lai [Lai94] and as a practical attack by Daemen et al. [DKR97]. Knudsen and Wagner formalized the concept [KW02]. The core idea of integral cryptanalysis is finding a set of inputs such that sum of the resulting outputs is key-independent in some positions. In the context of symmetric-key cryptography, any primitives can be represented as a vectorial Boolean function in a combination of secret and public variables. The set of inputs in bit-based integral analysis usually consists of inputs taking all possible combinations in d input bits, whereas the remaining bits take a fixed value. Such input sets form a linear subspace of dimension d, so the resulting output sets are the dth derivative of the corresponding vectorial Boolean function with respect to the active input bits, i.e., those bits that are not fixed [Lai94]. An approach to find the key-independent output positions is detecting those output bits whose ANF do not include monomials containing key bits as well as active input bits, since the dth derivative of such output bits with respect to the active bits is constant (zero-sum or one-sum).

However, in terms of the computational complexity, the vectorial Boolean functions corresponding to cryptographic primitives are usually very hard to construct directly, and hence cryptographers have to use indirect approaches to inspect the algebraic properties of vectorial Boolean functions. *Monomial prediction* [HSWW20] is a new technique to determine the presence of a particular monomial in the product of the coordinate functions of a vectorial Boolean function, when directly constructing it is computationally infeasible. This technique exploits the fact that a cryptographic vectorial Boolean function $\boldsymbol{f} : \mathbb{F}_2^{n_0} \to \mathbb{F}_2^{n_r}$ is usually a composition of several simpler vectorial Boolean functions $\boldsymbol{f}_i : \mathbb{F}_2^{n_{i-1}} \to \mathbb{F}_2^{n_i}$, such that $\boldsymbol{x}^{(i)} = \boldsymbol{f}_i\left(\boldsymbol{x}^{(i-1)}\right)$ for $1 \leq i \leq r$, i.e., $\boldsymbol{f} = \boldsymbol{f}_r \circ \cdots \circ \boldsymbol{f}_1$, where the algebraic representation of each smaller vectorial Boolean function $\boldsymbol{f}_i$ is available. Assuming that $\boldsymbol{x}^{(i)} = \boldsymbol{f}_i\left(\boldsymbol{x}^{(i-1)}\right)$, for any $\boldsymbol{u}^{(i)} \in \mathbb{F}_2^{n_i}$, we can describe $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ in variables $\boldsymbol{x}^{(i-1)}$ as follows:

$$\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) = \sum_{\pi_{\boldsymbol{u}^{(i-1)}}(\boldsymbol{x}^{(i-1)}) \to \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})} \pi_{\boldsymbol{u}^{(i-1)}}(\boldsymbol{x}^{(i-1)})$$

Accordingly, to follow the presence of monomials in a sequence of vectorial Boolean functions that are applied one after another, Hu et al. [HSWW20] defined the concept of monomial trails and proposed how to model them using MILP.

Definition 1 (Monomial Trail [HSWW20]). Let $\boldsymbol{x}^{(i)} = \boldsymbol{f}_i(\boldsymbol{x}^{(i-1)})$ for $1 \leq i \leq r$. A sequence of monomials $(\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}), \ldots, \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}), \ldots, \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}))$ is an r-round monomial trail connecting $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ with respect to the composition function $\boldsymbol{f} = \boldsymbol{f}_r(\boldsymbol{x}^{(r-1)}) \circ \ldots \circ \boldsymbol{f}_1(\boldsymbol{x}^{(0)})$ if

$$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \cdots \to \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) \to \cdots \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}).$$

If there is at least one monomial trail from $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, we write $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Otherwise, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$.

For a given composition sequence of vectorial Boolean functions, the following lemma guarantees the absence of a monomial in the ANF of the resulted composite function.

Lemma 1 ([HSWW20]). *If* $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$*, then* $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$*; or, equivalently,* $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ *implies* $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \nrightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$.

Hu et al. [HSWW20] also show that the parity of the number of monomial trails connecting $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(\boldsymbol{r})}}(\boldsymbol{x}^{(r)})$ perfectly determines the presence or absence of $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ in the ANF of $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. This requires counting the monomial trails. Moreover, Hu et al. prove that the monomial prediction technique and the 3-Subset Division Property without Unknown subset (3SDPwoU) are equivalent and perfect detection algorithms to detect the presence or absence of monomials in the ANF of Boolean functions. The downside is that for composite vectorial Boolean functions coming from cryptographic primitives, the number of monomial trails between two monomials can be extremely large, which makes counting the number of monomial trails computationally difficult. However, as we will show in the next sections, Lemma 1 is sufficient to detect key-independent bits in integral cryptanalysis.

Key-recovery for integral attacks. The easiest way to extend an integral distinguisher where some bit positions sum to 0 across N queries to a key-recovery attack is to evaluate the sum for each of the 2^k key candidates, if k key bits are necessary to partially decrypt the ciphertext to the relevant target bits. Several techniques have been proposed to reduce this key-recovery complexity of $\mathcal{O}(N \cdot 2^k)$ partial decryptions under certain circumstances, including the partial-sum technique by Ferguson et al. [FKL+00], the Meet-in-the-Middle (MitM) technique by Sasaki and Wang [SW12], and the Fast Fourier Transform (FFT) technique by Todo and Aoki [TA14]. We refer to Subsection 4.2 for more details.

3 Modeling Monomial Prediction for Block Ciphers

Denoting the key, plaintext and ciphertext variables by $\boldsymbol{k}, \boldsymbol{x}$, and $\boldsymbol{y}$ respectively, a block cipher $\boldsymbol{y} = E(\boldsymbol{k}, \boldsymbol{x})$ can be considered as a family of functions indexed by $\boldsymbol{k}$:

$$\begin{aligned} E_{\boldsymbol{k}} &: \mathbb{F}_2^n \to \mathbb{F}_2^n \\ &(\boldsymbol{k}, \boldsymbol{x}) \mapsto \boldsymbol{y} = E(\boldsymbol{k}, \boldsymbol{x}). \end{aligned}$$

Hence, any product of ciphertext bits $\boldsymbol{y}^{\boldsymbol{v}}$ can be expressed as a Boolean function $f(\boldsymbol{k}, \boldsymbol{x})$:

$$\sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathbb{F}_2^n \times \mathbb{F}_2^k} a_{\boldsymbol{u},\boldsymbol{v}} \cdot \boldsymbol{k}^{\boldsymbol{v}} \boldsymbol{x}^{\boldsymbol{u}} = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} \left(\sum_{\boldsymbol{v} \in \mathbb{F}_2^k} a_{\boldsymbol{u},\boldsymbol{v}} \cdot \boldsymbol{k}^{\boldsymbol{v}} \right) \cdot \boldsymbol{x}^{\boldsymbol{u}} = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}}(\boldsymbol{k}) \cdot \boldsymbol{x}^{\boldsymbol{u}}, \tag{1}$$

where k denotes the key size, $a_{\boldsymbol{u},\boldsymbol{v}} \in \mathbb{F}_2$, for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_2^n \times \mathbb{F}_2^k$, and $a_{\boldsymbol{u}}(\boldsymbol{k}) = \sum_{\boldsymbol{v} \in \mathbb{F}_2^k} a_{\boldsymbol{u},\boldsymbol{v}} \cdot \boldsymbol{k}^{\boldsymbol{v}}$. In the single-key setting, $\boldsymbol{x}$ is a public variable, whereas $\boldsymbol{k}$ takes a fixed unknown value. For a fixed key $\boldsymbol{k}$ and for any $\boldsymbol{u} \in \mathbb{F}_2^n$, the coefficient of $\boldsymbol{x}^{\boldsymbol{u}}$ in Equation 1 is determined by

$$a_{\boldsymbol{u}}(\boldsymbol{k}) = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} f(\boldsymbol{k}, \boldsymbol{x}).$$

If $\pi_{\boldsymbol{v}}(\boldsymbol{k}) \cdot \pi_{\boldsymbol{u}}(\boldsymbol{x}) \not\rightsquigarrow f(\boldsymbol{k}, \boldsymbol{x})$ for all $\boldsymbol{v} \in \mathbb{F}_2^k$, then according to Lemma 1, $a_{\boldsymbol{u},\boldsymbol{v}} = 0$ for all $\boldsymbol{v} \in \mathbb{F}_2^k$, and hence $a_{\boldsymbol{u}}(\boldsymbol{k}) = 0$ for any $\boldsymbol{k} \in \mathbb{F}_2^k$. Moreover, if $\pi_{\boldsymbol{v}}(\boldsymbol{k}) \cdot \pi_{\boldsymbol{u}}(\boldsymbol{x}) \not\rightsquigarrow f(\boldsymbol{k}, \boldsymbol{x})$ holds for all $\boldsymbol{v} \in \mathbb{F}_2^k \setminus \{\boldsymbol{0}\}$, then $a_{\boldsymbol{u},\boldsymbol{v}} = 0$ for all $\boldsymbol{v} \in \mathbb{F}_2^k \setminus \{\boldsymbol{0}\}$, and as a result, $a_{\boldsymbol{u}}(\boldsymbol{k})$ is still key-independent, though it may not be zero. Given that an attacker can compute $a_{\boldsymbol{u}}(\boldsymbol{k}) = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} f(\boldsymbol{k}, \boldsymbol{x})$ by querying the encryption of $\mathbb{C}_{\boldsymbol{u}} = \{\boldsymbol{x} \in \mathbb{F}_2^n : \boldsymbol{x} \preceq \boldsymbol{u}\}$ in the chosen plaintext scenario, $a_{\boldsymbol{u}}(\boldsymbol{k})$ yields a distinguisher when it is key-independent. In addition, if $\pi_{\boldsymbol{v}}(\boldsymbol{k}) \cdot \pi_{\boldsymbol{w}}(\boldsymbol{x}) \not\rightsquigarrow f(\boldsymbol{k}, \boldsymbol{x})$, for all $\boldsymbol{w} \succeq \boldsymbol{u}$ and for all $\boldsymbol{v} \in \mathbb{F}_2^k$, then $a_{\boldsymbol{w}}(\boldsymbol{k})$ is key-independent for all $\boldsymbol{w} \succeq \boldsymbol{u}$, and hence $\sum_{\boldsymbol{x} \in \mathbb{C}_{\boldsymbol{u}}} f(\boldsymbol{k}, \boldsymbol{x})$ is key-independent, where $\mathbb{C}_{\boldsymbol{u}}$ is a set of plaintexts such that variables in $\{x_i : u_i = 1\}$ are taking all possible combinations and the remaining variables are fixed to some arbitrary values.

3.1 CP Encoding of Monomial Prediction

Every cryptographic primitive can be divided into some smaller vectorial Boolean functions whose ANF are available. Therefore, modeling the propagation of monomial trails through the building blocks of cryptographic primitives, such as S-box, Xor, and Copy, or generally small Boolean functions is sufficient to model the whole primitive. To model the behavior of a small vectorial Boolean function in terms of the propagation of monomial trails, we define the *monomial prediction table* (MPT).

Definition 2 (Monomial Prediction Table (MPT))**.** For a vectorial Boolean function $\boldsymbol{f} : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ with $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, the monomial prediction table (MPT) of $\boldsymbol{f}$ is a $2^m \times 2^n$ array such that for any $u \in \mathbb{F}_2^m$ and $v \in \mathbb{F}_2^n$, $\mathrm{MPT}(u, v) = 1$ if $\pi_{\boldsymbol{u}}(\boldsymbol{x}) \rightarrow \pi_{\boldsymbol{v}}(\boldsymbol{y})$, and $\mathrm{MPT}(u, v) = 0$ otherwise.

Table 5: Monomial Prediction Table (MPT) of `WARP`'s S-box.

$\boldsymbol{u} \setminus \boldsymbol{v}$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	1	·	·	·	1	·	·	·	1	·	·	·	1	·	·	·
1	·	·	1	·	1	·	·	·	·	·	1	·	1	·	·	·
2	·	1	·	·	·	1	·	·	·	1	·	·	·	1	·	·
3	·	·	·	1	·	1	·	·	1	1	1	·	·	1	·	·
4	·	·	1	·	·	·	1	·	·	·	1	·	·	·	1	·
5	·	1	1	1	·	·	1	·	·	1	1	1	·	·	1	·
6	·	·	·	1	·	·	·	1	·	·	·	1	·	·	·	1
7	·	1	·	·	1	1	1	·	·	1	·	·	·	·	·	1
8	·	·	·	·	1	·	·	·	·	·	·	·	1	·	·	·
9	·	1	1	·	1	·	·	·	·	1	1	·	1	·	·	·
a	·	·	·	·	·	1	·	·	1	1	·	·	·	1	·	·
b	·	1	·	1	1	·	·	·	1	·	1	·	·	1	·	·
c	·	·	1	·	·	·	1	·	1	·	1	·	·	·	1	·
d	·	·	·	1	·	·	1	·	·	·	1	1	·	·	1	·
e	·	1	·	1	1	·	·	1	1	·	·	1	·	·	·	1
f	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1

Application to `WARP`'s S-box. Table 5 represents the MPT of `WARP`'s S-box. Given that $\mathrm{MPT}(\boldsymbol{u}, \boldsymbol{v})$ can be interpreted as a Boolean function in variables $(\boldsymbol{u}, \boldsymbol{v})$, we can utilize the Quine-McCluskey [Qui52, McC56] or Espresso [BHMSV84] algorithms to minimize its conjunctive normal form (CNF) in our CP and SAT models. For instance, all valid monomial propagations $(u_0, u_1, u_2, u_3) \rightarrow (v_0, v_1, v_2, v_3)$ through the `WARP`'s S-box can be encoded as the satisfying solutions of the following CNF, where u_0 and v_0 correspond to the MSB of input and output, respectively:

$$\begin{array}{lll}
\ (u_2 \vee \neg v_1 \vee \neg v_3) & \wedge\ (\neg u_1 \vee \neg v_0 \vee \neg v_1 \vee v_2) & \wedge\ (\neg u_0 \vee \neg u_1 \vee \neg u_2 \vee \neg v_2 \vee v_3) \\
\wedge\ (u_2 \vee u_3 \vee \neg v_3) & \wedge\ (\neg u_0 \vee \neg u_1 \vee \neg u_3 \vee v_2) & \wedge\ (\neg u_0 \vee \neg u_3 \vee v_0 \vee \neg v_1 \vee \neg v_3) \\
\wedge\ (u_1 \vee \neg v_1 \vee \neg v_2) & \wedge\ (\neg u_1 \vee u_2 \vee v_0 \vee v_2 \vee v_3) & \wedge\ (\neg u_0 \vee \neg u_1 \vee \neg u_3 \vee v_0 \vee v_1 \vee v_3) \\
\wedge\ (u_1 \vee u_3 \vee \neg v_2) & \wedge\ (u_2 \vee \neg u_3 \vee v_1 \vee v_2 \vee v_3) & \wedge\ (\neg u_0 \vee \neg u_2 \vee \neg u_3 \vee \neg v_0 \vee v_1 \vee \neg v_3) \\
\wedge\ (u_0 \vee \neg u_2 \vee u_3 \vee v_3) & \wedge\ (u_1 \vee \neg v_0 \vee \neg v_2 \vee \neg v_3) & \wedge\ (\neg u_1 \vee \neg u_2 \vee \neg u_3 \vee v_1 \vee \neg v_2) \\
\wedge\ (u_0 \vee \neg u_1 \vee u_3 \vee v_2) & \wedge\ (\neg u_0 \vee u_1 \vee u_3 \vee v_0 \vee v_1) & \wedge\ (\neg u_1 \vee \neg u_2 \vee \neg u_3 \vee v_1 \vee v_3) \\
\wedge\ (\neg u_2 \vee v_0 \vee v_1 \vee v_3) & \wedge\ (\neg u_1 \vee u_3 \vee \neg v_0 \vee v_2 \vee \neg v_3) & \wedge\ (u_0 \vee u_1 \vee \neg u_3 \vee v_0 \vee v_1 \vee v_2) \\
\wedge\ (u_0 \vee u_1 \vee u_2 \vee \neg v_3) & \wedge\ (u_0 \vee u_1 \vee \neg u_2 \vee \neg v_1 \vee v_3) & \wedge\ (\neg u_3 \vee v_0 \vee \neg v_1 \vee \neg v_2 \vee \neg v_3) \\
\wedge\ (u_1 \vee u_2 \vee \neg v_2 \vee \neg v_3) & \wedge\ (u_1 \vee \neg u_2 \vee u_3 \vee \neg v_1 \vee v_3) & \wedge\ (\neg u_0 \vee u_1 \vee u_2 \vee v_1 \vee v_2 \vee v_3) \\
\wedge\ (\neg u_2 \vee \neg v_0 \vee \neg v_1 \vee v_3) & \wedge\ (\neg u_1 \vee u_3 \vee \neg v_1 \vee v_2 \vee \neg v_3). &
\end{array}$$

General propagation rules. The propagation rules for monomial prediction of other building blocks of block ciphers and their CP/SAT encoding are summarized as follows.

Proposition 1 (AND or Multiplication in $\mathbb{F}_2$)**.** *. For $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, $f(x_0, x_1, \ldots, x_{n-1}) = x_0 \cdot x_1 \cdots x_{n-1}$, all valid monomial propagation $(u_0, \ldots, u_{n-1}) \rightarrow (v)$ can be encoded as:*

$$\left(\bigwedge_{i=0}^{n-1} (v \vee \neg u_i)\right) \wedge \left(\bigwedge_{i=0}^{n-1} (\neg v \vee u_i)\right).$$

Proposition 2 (2-bit XOR or Addition in $\mathbb{F}_2$)**.** *For $f : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$, $f(x_0, x_1) = x_0 \oplus x_1$, all valid monomial propagation $(u_0, u_1) \rightarrow (v)$ can be encoded as*

$$(\neg u_0 \vee \neg u_1) \wedge (\neg u_0 \vee v) \wedge (\neg u_1 \vee v) \wedge (u_0 \vee u_1 \vee \neg v).$$

Proposition 3 (3-bit XOR)**.** *For $f : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2$, $f(x_0, x_1, x_2) = x_0 \oplus x_1 \oplus x_2$, all valid monomial propagation $(u_0, u_1, u_2) \rightarrow (v)$ through f can be encoded as:*

$$(\neg u_0 \vee \neg u_1) \wedge (\neg u_0 \vee \neg u_2) \wedge (\neg u_1 \vee \neg u_2) \wedge (\neg u_0 \vee v) \wedge (\neg u_1 \vee v) \wedge (\neg u_2 \vee v) \wedge (u_0 \vee u_1 \vee u_2 \vee \neg v).$$

Proposition 4 (Negation of 3-bit XOR)**.** *For $f : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2$, $f(x_0, x_1, x_2) = x_0 \oplus x_1 \oplus x_2 \oplus 1$, the valid monomial propagation $(u_0, u_1, u_1) \rightarrow (v)$ through f can be encoded as:*

$$(\neg u_0 \vee \neg u_1) \wedge (\neg u_0 \vee \neg u_2) \wedge (\neg u_1 \vee \neg u_2) \wedge (\neg u_1 \vee v) \wedge (\neg u_2 \vee v) \wedge (\neg u_0 \wedge v).$$

Proposition 5 (Branching Point or COPY)**.** *For $\boldsymbol{f} : \mathbb{F}_2 \rightarrow \mathbb{F}_2^n$, $\boldsymbol{f}(x) = (x, x, \ldots, x)$, all valid monomial propagation $(u) \rightarrow (v_0, \ldots, v_{n-1})$ through $\boldsymbol{f}$ can be encoded as:*

$$u = \bigvee_{i=0}^{n-1} v_i.$$

Model for `WARP`. To denote the monomials corresponding to the input of the $i + 1$th round, we use $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$. Accordingly, assuming that we are analyzing r rounds of `WARP`, $\pi_{\boldsymbol{u}^0}(\boldsymbol{x}^{(0)})$, and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ represent the monomials corresponding to the plaintext and ciphertext, respectively, where $1 \leq r \leq 41$. To model the key schedule of `WARP`, we denote by $\pi_{\boldsymbol{v}^{(i)}}(\boldsymbol{k}^{(i)})$ the round key of the ith round, and $\pi_{\boldsymbol{v}^{(0)}}(\boldsymbol{k}^{(0)})$ represents the monomials in variables of the master key $\boldsymbol{k}$.

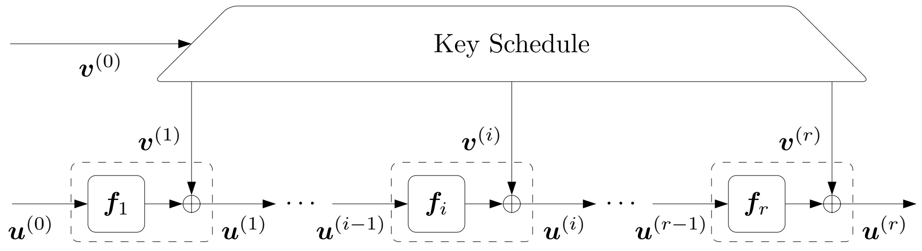

Figure 2: Main variables in our CP model for block ciphers

Given that a monomial $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) \cdot \pi_{\boldsymbol{v}^{(i)}}(\boldsymbol{k}^{(i)})$ is uniquely specified by $\boldsymbol{u}^{(i)}$ and $\boldsymbol{v}^{(i)}$, as illustrated in Figure 2, our CP and SAT models are described based on variables in $\boldsymbol{u}^{(i)}$ and $\boldsymbol{v}^{(i)}$, for $0 \leq i \leq r$. In addition to the main variables represented in Figure 2, we define some additional variables corresponding to the input and output of S-boxes as well. Next, using the propagation rules for S-box, COPY, and XOR with constant, we link the variables of our model to encode the propagation of monomial trails through r rounds of `WARP`, taking the key schedule into account. As a result, any feasible solution of the constructed CP model corresponds to a valid monomial trail. To check the absence of a monomial $\pi_{\boldsymbol{v}}(\boldsymbol{k}^{(0)}) \cdot \pi_{\boldsymbol{u}}(\boldsymbol{x}^{(0)})$ for a certain $\boldsymbol{u} \in \mathbb{F}_2^n$ and any possible values of $\boldsymbol{v} \in \mathbb{F}_2^k$ in the ANF of the ith output bit, we check the satisfiability of the model, where $\boldsymbol{u}^{(0)} = \boldsymbol{u}$ and $\boldsymbol{u}^{(r)} = \boldsymbol{e}_i$. If the model becomes unsatisfiable, then there is no valid monomial trail starting from $\pi_{\boldsymbol{v}^{(0)}}(\boldsymbol{k}^{(0)}) \cdot \pi_{\boldsymbol{u}}(\boldsymbol{x}^{(0)})$, which guarantees that the ith output bit is balanced over the set of inputs $\mathbb{C}_{\boldsymbol{u}} := \{\boldsymbol{x} \in \mathbb{F}_2^n : \boldsymbol{x} \leq \boldsymbol{u}\}$, according to Lemma 1.

If we only fix the variables in $\boldsymbol{u}^{(0)}$ corresponding to the nonzero values of $\boldsymbol{u}$ and let the rest of its variables be free, then the unsatisfiality of the model guarantees the absence of $\pi_{\boldsymbol{v}}(\boldsymbol{k}^{(0)}) \cdot \pi_{\boldsymbol{w}}(\boldsymbol{x}^{(0)})$ in the ANF of the ith output bit for all $\boldsymbol{v} \in \mathbb{F}_2^k$ and for all $\boldsymbol{w} \geq \boldsymbol{u}$. The advantage is that the constant part of the resulting cube of plaintexts over which the ith output bit is balanced should not necessarily take zero value, which gives us more freedom to adjust the overall complexity of the resulting integral attack. Moreover, we can include the constraint $\boldsymbol{v}^{(0)} \neq \boldsymbol{0}$ to check whether the ith output bit is key-independent, and detect the one-sum property as well.

We implemented the automatic method to search for integral distinguishers based on monomial prediction with three popular encoding methods: CP, MILP, and SAT. It is worth noting that, during our experiments, SAT encoding resulted in a much better performance than the MILP or CP encoding methods. Our approach to taking the key schedule into account is applicable for linear and nonlinear key schedules. However, the efficiency of solving the resulting model may decrease where the key schedule involves many basic operations.

3.2 Improved Integral Distinguishers of WARP

The best previous integral distinguisher for WARP was reported by its designers [BBI+20], who propose a 20-round integral distinguisher discovered by a nibble-wise model based on 2-Subset Division Property (2SDP). The designers of WARP also claimed that finding an integral distinguisher with a lower data complexity is only possible for less than 20 rounds. Here, thanks to the higher accuracy of our bit-wise model based on monomial prediction to search for integral distinguishers, we not only find a 20-round integral distinguisher with lower data complexity and more balanced output bits, but we are also able to find integral distinguishers for up to 22 rounds of WARP. Exploiting the intrinsic properties of WARP, we also prove that the distinguishers derived from our automatic search can be extended by one round further at the end as well as the beginning. As a result, we can improve the integral distinguishers of WARP by 4 rounds in total.

To look for the longest integral distinguishers of WARP, we check all of the 128 possible plaintext structures with only one constant bit to see whether they yield a key-independent bit at the output. Accordingly, we discovered that for 26 out of 128 possible positions for the constant bit at the input, there is at least one output bit with zero-sum property after 22 rounds. Setting bits (2) or (66) to a constant value results in the highest number of balanced bits at the following output positions:

$$(2) \xrightarrow{\text{22 rounds}} (\underline{20, 21, 22, 23},\ 118,\ \underline{60, 61, 62, 63}), \tag{2}$$

$$(66) \xrightarrow{\text{22 rounds}} (54,\ \underline{84, 85, 86, 87},\ \underline{124, 125, 126, 127}). \tag{3}$$

To find an integral distinguisher with a lower data complexity, we also implemented the method introduced by Eskandari et al. [EKKT18] to minimize the number of active bits taking the one-sum property into account as well. Consequently, we verified that there is no integral distinguisher with a lower data complexity for 22 rounds. Next, applying the same strategy for 21 rounds, we found the following distinguishers which are optimal for 21 rounds in terms of data complexity:

$$(\underline{24, 25, 26, 27}) \xrightarrow{\text{21 rounds}} (22), \quad (\underline{88, 89, 90, 91}) \xrightarrow{\text{21 rounds}} (86). \tag{4}$$

For 20 rounds, we discovered a distinguisher with one constant nibble at the input yielding 13 more output bits with zero-sum property in comparison to the integral distinguisher proposed by the designers:

$$(\underline{64,65,66,67}) \xrightarrow{\text{20 rounds}} (6, \underline{20,21,22,23}, \underline{52,53,54,55}, \underline{60,61,62,63}, \underline{76,77,78,79},$$
$$\underline{84,85,86,87}, \underline{116,117,118,119}, \underline{124,125,126,127}).$$

To disprove the implicit claim of the designers regarding the nonexistence of a 20-round integral distinguisher with data complexity lower than 2^{124} chosen plaintexts, we provide the following 20-round distinguishers with a data complexity of 2^{123} chosen plaintexts:

$$(\underline{24,25,26,27}, i) \xrightarrow{\text{20 rounds}} (\underline{20,21,22,23}, \underline{60,61,62,63}), \quad \text{for} \quad 28 \leq i \leq 31. \tag{5}$$

Extension to a generalized integral distinguisher. The designers of WARP put the Xor with sub-key operation after the S-box application to avoid the complementary property of Feistel-type structure. However, as shown in the following theorem, this property lets us extend all of our discovered integral distinguishers by one round further.

Theorem 1. *Any integral distinguisher for WARP built upon a multiset of even size which yields at least one key-independent bit after r rounds, can be extended to an $r+1$-round generalized integral distinguisher with the same data complexity.*

Proof. Let $\mathbb{C}$ denote the multiset of input plaintexts. If the jth bit of $\sum_{\mathbb{C}} X_{2i}^{(r)}$ is key-independent for some $0 \leq i \leq 15$, then the same position in $\sum_{\mathbb{C}} X_{\pi(2i)}^{(r+1)}$ must be key-independent as well, since $X_{\pi(2i)}^{(r+1)} = X_{2i}^{(r)}$. If the jth bit $\sum_{\mathbb{C}} X_{2i+1}^{(r)}$ is key-independent for some $0 \leq i \leq 15$, then the jth bit of $\sum_{\mathbb{C}}(S(X_{\pi(2i)}^{(r+1)}) \oplus X_{\pi(2i+1)}^{(r+1)})$ is key-independent since we have:

$$\sum_{\mathbb{C}} X_{2i+1}^{(r)} = \sum_{\mathbb{C}} \left(S(X_{\pi(2i)}^{(r+1)}) \oplus X_{\pi(2i+1)}^{(r+1)} \right) \oplus \sum_{\mathbb{C}} K_i^{(b)},$$

and $\sum_{\mathbb{C}} K_i^{(b)} = 0$ for any fixed master key, where $b \in \{0,1\}$. □

In Subsection 4.1, we show how to prepend another initial round to the distinguishers.

Experimental verification. To experimentally validate the outcomes of our automatic tool to search for integral distinguishers, we also discovered some practical integral distinguishers with very low data complexity for up to 15 rounds of WARP which are summarized in Table 6. The very low data complexity of these distinguishers makes anyone able to evaluate their correctness with very limited computational resources.

Table 6: Practical integral distinguishers for 10 to 15 rounds of WARP.

#Rounds	Active Input Nibbles	Output Nibbles with Zero-Sum Property
10	(15)	(0,1,3,7,9,10,12,15,16,18,19,20,25,27,29,31)
11	(15)	(13,15,17,25,31)
12	(0,1)	(1,9,15,29,31)
13	(4,5,11)	(1,9,15,29,31)
14	(6,7,8,9,29)	(13,15,17,25,31)
15	(1,2,3,7,28,29,30,31)	(13,15,17,25,31)

4 Key-Recovery Attack Based on Integral Distinguishers

We now extend our 22-round integral distinguisher from Equation 2 to a 32-round key-recovery attack by prepending 1 initial round and appending 9 final rounds. The alternative 22-round distinguisher from Equation 3 could be similarly extended.

4.1 Prepending 1 Round

We first prepend an initial round to the 22-round integral distinguisher from Equation 2. This distinguisher requires an input set of all 2^{127} chosen plaintexts $X^{(1)}$ where bit 2 in nibble $X_0^{(1)}$ is constant. Let $\mathbb{L}_c = \{(x_0, x_1, x_2, x_3) \in \mathbb{F}_2^4 \mid x_2 = c\}$ denote the set of suitable values for $X_0^{(1)}$ with constant c. If we prepend one round, this nibble is computed as

$$X_0^{(1)} = S(X_{10}^{(0)}) \oplus X_{11}^{(0)} \oplus K_5^{(0)},$$

where $K_5^{(0)}$ is constant, as illustrated in Figure 3. Thus, we get $X_0^{(1)} \in \mathbb{L}_c$ with c equal to the constant bit 2 of $K_5^{(0)}$ if we choose the input set $\mathbb{S}$ with 2^{127} chosen plaintexts:

$$\mathbb{S} = \{X^{(0)} \in \mathbb{F}_2^{128} \mid S(X_{10}^{(0)}) \oplus X_{11}^{(0)} \in \mathbb{L}_0\}.$$

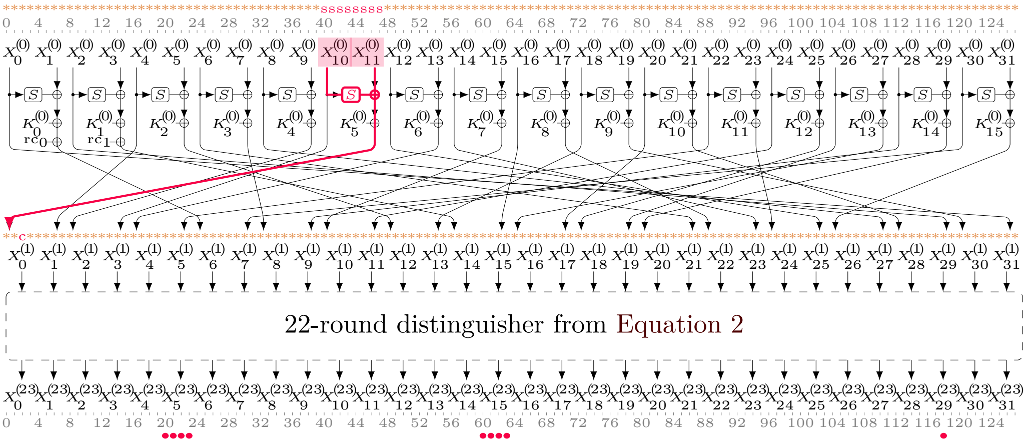

Figure 3: Prepending 1 initial round to the 22-round integral distinguisher from Equation 2 to obtain a 23-round distinguisher with data complexity 2^{127} and 9 zero-sum bits in $X^{(23)}$.

Combined with the extension of Theorem 1, we obtain a generalized, set-based integral distinguisher for $1+22+1 = 24$ rounds with the same data complexity of 2^{127}. To the best of our knowledge, this is the best distinguisher for `WARP` so far. The approach can similarly be applied to the 21-round distinguisher with data complexity 2^{124} in Equation 4 and the 20-round distinguisher with 2^{123} in Equation 5 to obtain distinguishers for $1+21+1 = 23$ and $1+20+1 = 22$ rounds, respectively, with the previous complexities.

4.2 Cost Model for Appending Key-Recovery Rounds

The $1+22$-round distinguisher of Figure 3 gives us 9 zero-sum bits at the output of 2^{127} queried plaintexts. In the design document [BBI$^+$20, Appendix B.3], the designers discuss a 20-round integral distinguisher and claim that due to the high data complexity, a key-recovery attack can only extend the distinguisher by at most one round and the resulting complexity is "almost close to an exhaustive key search". However, this estimate appears to assume a straightforward key-recovery approach where each of the queried plaintexts is partially decrypted under each of the key guesses, which would require 2^{124+4} 1-round decryptions in their example[1].

[1] We remark that due to the effect described in Theorem 1, a 1-round key-recovery attack actually cannot recover any key material, as the key simply cancels out.

MitM FFT key-recovery approach. With more advanced approaches for integral attacks such as the Meet-in-the-Middle (MitM) technique [SW12] and the Fast Fourier Transform (FFT) technique [TA14], as well as a dedicated analysis exploiting `WARP`'s properties such as the position of key addition, we can cover substantially more rounds. Consider a target nibble $\bigoplus_{\mathbb{C}} X_i^{(r)} = 0$ with i odd for an r-round integral distinguisher with an input structure $\mathbb{C}$ of even size $|\mathbb{C}|$. When we want to recover this nibble based on some key guess and a given set of ciphertexts, `WARP`'s Feistel structure permits us to use the Meet-in-the-Middle approach and independently recover its two contributing branches as

$$\bigoplus_{\mathbb{C}} X_i^{(r)} = \bigoplus_{\mathbb{C}} S(X_{\pi(i-1)}^{(r+1)}) \oplus X_{\pi(i)}^{(r+1)} \oplus K_{(i-1)/2}^{(r \bmod 2)} = \bigoplus_{\mathbb{C}} S(X_{\pi(i-1)}^{(r+1)}) \oplus \bigoplus_{\mathbb{C}} X_{\pi(i)}^{(r+1)} \overset{!}{=} 0\,,$$

where the key sum $\bigoplus_{\mathbb{C}} K_{(i-1)/2}^{(r \bmod 2)} = 0$ cancels out for an even-sized set $\mathbb{C}$. We refer to these two nibbles $X_{\mathrm{L}} = S(X_{\pi(i-1)}^{(r+1)})$ and $X_{\mathrm{R}} = X_{\pi(i)}^{(r+1)} = X_{\pi(\pi(i))}^{(r+2)}$ as the left and right target nibbles, respectively. For both nibbles $X_{\mathrm{L}}, X_{\mathrm{R}}$ and each of their 2^4 possible values $\bigoplus_{\mathbb{C}} X_{\mathrm{L}}$, $\bigoplus_{\mathbb{C}} X_{\mathrm{R}}$, we want to construct a list of key candidates that generates this value. By matching these two lists based on the value (and potentially consistency checks between the two partial key candidates), we obtain all combined key guesses that satisfy the integral property $\bigoplus_{\mathbb{C}} X_i^{(r)} = 0$. For a 4-bit property and a 128-bit involved key, we expect a list of 2^{124} remaining key candidates that we can then check by brute-force evaluation. Note that we could also take additional integral conditions into account before brute-forcing (in our distinguishers, we have 9 rather than just 4 zero-sum bits); however, this won't substantially improve the overall attack complexity, so for simplicity, we focus on just one target nibble with its two branches X_{L} and X_{R} in the following.

To recover each of these branches, we need to compute the targeted nibbles as a function of the ciphertext and key nibbles. Ideally, in case we can represent the target nibble as a function $X = F(\tilde{K} \oplus \tilde{C})$ that depends only on the Xor of a partial k-bit ciphertext $\tilde{C}$ and key $\tilde{K}$, then we can recover $\bigoplus_{\mathbb{C}} X$ with a complexity of $\mathcal{O}(k \cdot 2^k)$ using the FFT technique for integral attacks [TA14], plus the initial cost of $|\mathbb{C}|$ queries. As the FFT approach works with Boolean functions, we actually need to repeat this 4 times in parallel for the 4 bits of the target nibble; in the following, $f(\cdot) : \mathbb{F}_2^k \to \mathbb{F}_2$ refers to a Boolean coordinate function of $F(\cdot) : \mathbb{F}_2^k \to \mathbb{F}_2^4$. More specifically, the FFT approach can compute $u_{\tilde{K}} = \sum_{C \in \mathbb{C}} f(\tilde{C} \oplus \tilde{K})$ for all $\tilde{K}$ with the help of the Fast Walsh-Hadamard Transform (FWHT) and two k-dimensional vectors v (encoding f) and w (encoding the set $\mathbb{C}$):

$$v_i = f(i) \in \{0,1\}, \qquad w_i = \#\{C \in \mathbb{C} \mid \tilde{C} = i\} \bmod 2 \in \{0,1\}, \qquad 0 \le i < 2^k.$$

Here, $w_i \in \{0,1\}$ counts the number of appearances of the partial ciphertext value i in $\mathbb{C}$, where we first eliminate all duplicate appearances as they cancel out in $\bigoplus_{\mathbb{C}} f(\cdot)$. The results $u_{\tilde{K}}$ are computed using the 2^k-dimensional Walsh matrix $\mathbf{H}_{2^k}$ as

$$u = \mathbf{V} \times w = \frac{1}{2^k} \mathbf{H}_{2^k} \times \mathrm{diag}(\mathbf{H}_{2^k} v) \times \mathbf{H}_{2^k} w$$

using $4 \cdot k \cdot 2^k$ additions and can be converted to the desired integral sum $\bigoplus_{\mathbb{C}} f(\tilde{C} \oplus \tilde{K})$ by reduction modulo 2 [TA14].

Dependency DAG for `WARP`. To find the function $F(\cdot)$ and thus evaluate the cost of recovering each of these branches, we construct a directed acyclic graph (DAG), the *dependency DAG*, to compute the targeted nibbles as a function of the ciphertext and key nibbles. Each node of the DAG represents a nibble of the internal state or key of `WARP`. A directed edge from one node (parent) to another (child) indicates that the parent node depends on the child node. Here, we can take advantage of the properties of `WARP` to compress the DAG and thus the input bitsize k of F.

- **Nodes** in `WARP` have one of the following types:
  - *Temporary* (`T`): any internal state nibble $X_i^{(r)}$ starts as a temporary node and can be expanded into one of the following types based on the cipher definition. In particular, a node $X_i^{(r)}$ turns either into a `X` node (if i odd and $r \leq R-1$, spawning children for the left and right cells $X_{\pi(i-1)}^{(r+1)}, X_{\pi(i)}^{(r+1)}$ and the key $K_{(i-1)/2}^{(r \bmod 2)}$; or similarly if i even and $r \leq R-2$ via $X_{\pi(i)}^{(r+1)}$) or into a `C` node (otherwise).
  - *Xor* (`X`): a nibble that is computed as the Xor of its children.
  - *Key* (`K`): either a key nibble $K_i^{(r)}$ or a *synthetic* key nibble corresponding to the Xor of several key nibbles.
  - *Ciphertext* (`C`): either a ciphertext nibble $C_i^{(R)} = X_i^{(R)}$ or a *synthetic* ciphertext nibble $S(C_{i-1}^{(R)}) \oplus C_i^{(R)}$. In particular, a temporary node $X_i^{(R-1)}$ is expanded into a `C`-node $C_i^{(R)}$ if i is even, or into an `X`-node with two children: a `C`-node $S(C_{i-1}^{(R)}) \oplus C_i^{(R)}$ and a `K`-node.
- **Edges** in `WARP` always link children with a parent of type `X`, so the parent is computed as the Xor-sum of (a function of) its children. The dependency can be of two types, representing the function applied to the child before Xoring:
  - *Identity* (`I`) uses the child directly.
  - *S-box* (`S`) uses S(child).

When constructing the DAG, starting from the target nibble as the root node, each node is first created as a temporary `T`-node and then expanded based on the definition of the cipher as described above. Each node is created based on a given nibble position and with a specified parent and edge type, with the following additional rules:

- If the parent of an `X`-node is another `X`-node, it is merged with its parent upon creation, and all the child node's children are directly attached to its parent.
- If the nibble is a key or ciphertext nibble, and the parent node already has a child (sibling) of type `K` or `C`, respectively, then no new node is created; instead, the sibling is converted to a synthetic nibble and merged with the given nibble.
  - If the sibling previously had *multiple parents*, it is split into two nodes and only the copy for the target parent is updated with the new cell.
  - Conversely, if the updated synthetic nibble now equals a previously created synthetic nibble, these two are merged into one node by adding the parent to the previously created nibble
- If the nibble already exists as a node, this node is reused, i.e., it has multiple parents.

We now want to point out some special properties of the `WARP` dependency DAG. Unlike typical Feistel constructions, `WARP` does not add the round keys at the beginning of each F-function, but at the end, together with the Feistel-Xor in an `X`-node. Additionally, the permutation after the F-function step will move each right (odd-indexed) branch to a left (even-indexed) and then back to a right branch, corresponding to another `X`-node. These `X`-nodes will be merged, and thus the corresponding keys will also be merged into synthetic key nibbles. This continues recursively until reaching the ciphertext. As a result, all involved keys in a path will be merged into a synthentic key nibble that will end up as a sibling to a synthetic ciphertext node. In other words, the resulting function F will be of a form $F(\tilde{K} \oplus \tilde{C})$ with the synthetic key nibbles $\tilde{K}$ and synthetic ciphertext nibbles $\tilde{C}$. If we consider the diffusion of the target nibble, then the number of distinct synthetic ciphertext nibbles equals the number of activated nibbles *before* the last round, in $X^{(R-1)}$

(based on the definition of C-nodes). The number of synthetic key nibbles and thus the value k determining the complexity may be higher, as some C-nodes may appear as siblings to multiple K-nodes. Still, we observed that the number of K-nodes in our construction is typically lower than a naive count of involved key nibbles. Additional optimizations can take the *rank* of the equivalent key into account, i.e., the rank of the matrix $\mathbf{M}$ such that $\tilde{K} = \mathbf{M} \cdot K$. We discuss this in more detail for the specific dependency DAGs in our attack in the following.

4.3 Key-Recovery Attack on 32 Rounds

Based on the previous discussion, the number of key-recovery rounds is essentially upper-bounded by the number of rounds required for full diffusion. WARP generally achieves full diffusion after 10 rounds; the following additional factors impact the number of rounds we can attack:

- Full diffusion in WARP is after 10 rounds for even-indexed branches, but 9 rounds for odd-indexed branches.
- For the MitM attack, the limit is full diffusion of the left branch X_L, which is an odd-indexed; but we gain the additional first round, where the target nibble is split into X_L and X_R.
- We require that diffusion is not full *before* the last round, thus gaining one round: in the last round, we can work with synthetic ciphertext nibbles.

Based on these limitations, we can hope to add no more than 10 rounds. Considering the potential target nibbles in the distinguishers of Equation 2 and 3, 10 rounds activate between 29 and 31 of the 32 output nibbles in the left branch X_L, but depend on all 128 bits of key material. The resulting complexity of FFT key recovery would thus probably be above the generic brute-force complexity, depending on the assumed conversion factor of the complexity of additions versus cipher evaluations and other details. We thus focus on 9 rounds appended to the 23-round distinguisher, or 32 rounds (out of 41) overall.

In the distinguisher of Equation 2 illustrated in Figure 3, we can choose one of two target nibbles, $X_5^{(23)}$ or $X_{15}^{(23)}$. Their properties are similar, but not identical: $X_{15}^{(23)}$ activates fewer nibbles before the last round and thus creates fewer synthetic ciphertext nodes (24 for $X_{15}^{(23)}$, 26 for $X_5^{(23)}$), but both require the same amount of key material. We focus on $X_5^{(23)}$ for simplicity.

Overall, the target nibble $X_5^{(23)}$ appears to depend on all ciphertext nibbles and all key nibbles. Applying our dependency algorithm to the right and left branches of the MitM approach in Figure 4, we obtain the dependency DAGs in Figure 5 and Figure 6, respectively. We can now use them to separately recover $\bigoplus_{\mathbb{C}} X_\mathrm{R} = \bigoplus_{\mathbb{C}} X_{12}^{(24)}$ (right 🟩) and $\bigoplus_{\mathbb{C}} X_\mathrm{L} = \bigoplus_{\mathbb{C}} S(X_1^{(24)})$ (left 🟦) bit-by-bit under each key guess and then merge the results to obtain $2^{128-4} = 2^{124}$ candidates for the full key. Among these, we can brute-force for the correct key.

Right branch: $\bigoplus_{\mathbb{C}} \boldsymbol{X}_\mathbf{R} = \bigoplus_{\mathbb{C}} \boldsymbol{X}_{\mathbf{12}}^{\mathbf{(24)}} = \bigoplus_{\mathbb{C}} \boldsymbol{F}_\mathbf{R}(\tilde{\boldsymbol{C}}_\mathbf{R}, \tilde{\boldsymbol{K}}_\mathbf{R})$. According to the dependency DAG illustrated in Figure 5, $X_{12}^{(24)}$ can be recovered as a function $F_\mathrm{R}(\tilde{K}_\mathrm{R}, \tilde{C}_\mathrm{R})$ using 84 bits (21 nibbles) of key information and 80 bits (20 distinct nibbles) of partial ciphertext, of which one appears twice. In a straightforward analysis based on Figure 4, on the other hand, the nibble X_R appears to depend on 27 distinct key nibbles and 26 ciphertext nibbles. Here, the key information corresponds to an equivalent key $\tilde{K}_\mathrm{R}$ computed as a linear function with full rank of the original key. Similarly, the ciphertext information corresponds to an equivalent ciphertext $\tilde{C}_\mathrm{R}$ whose nibbles are nonlinear combinations of the original ciphertext nibbles. We will recover the equivalent key based on the equivalent ciphertexts.

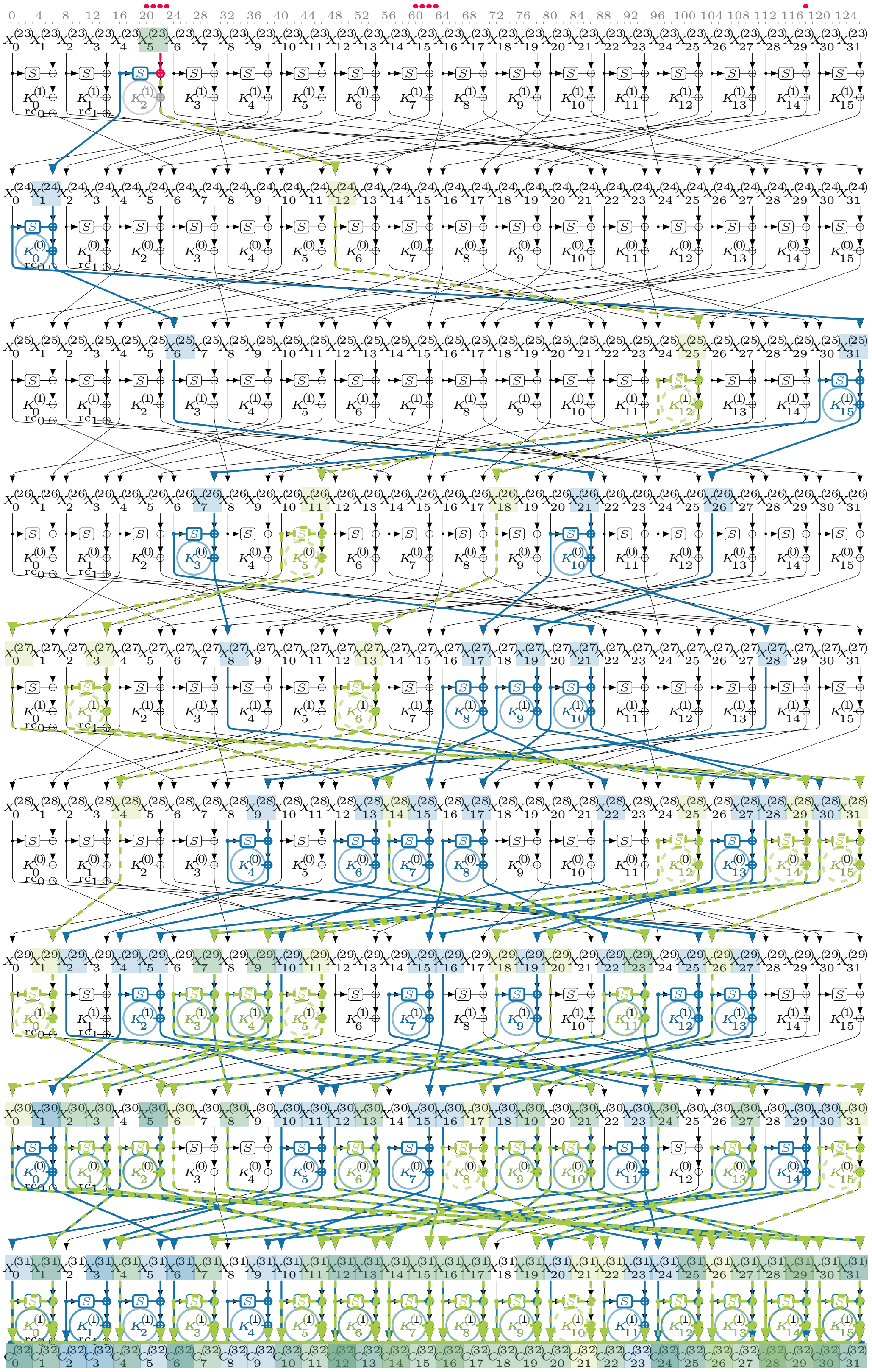

Figure 4: Key recovery for 9 rounds of WARP after the 23-round distinguisher of Figure 3.

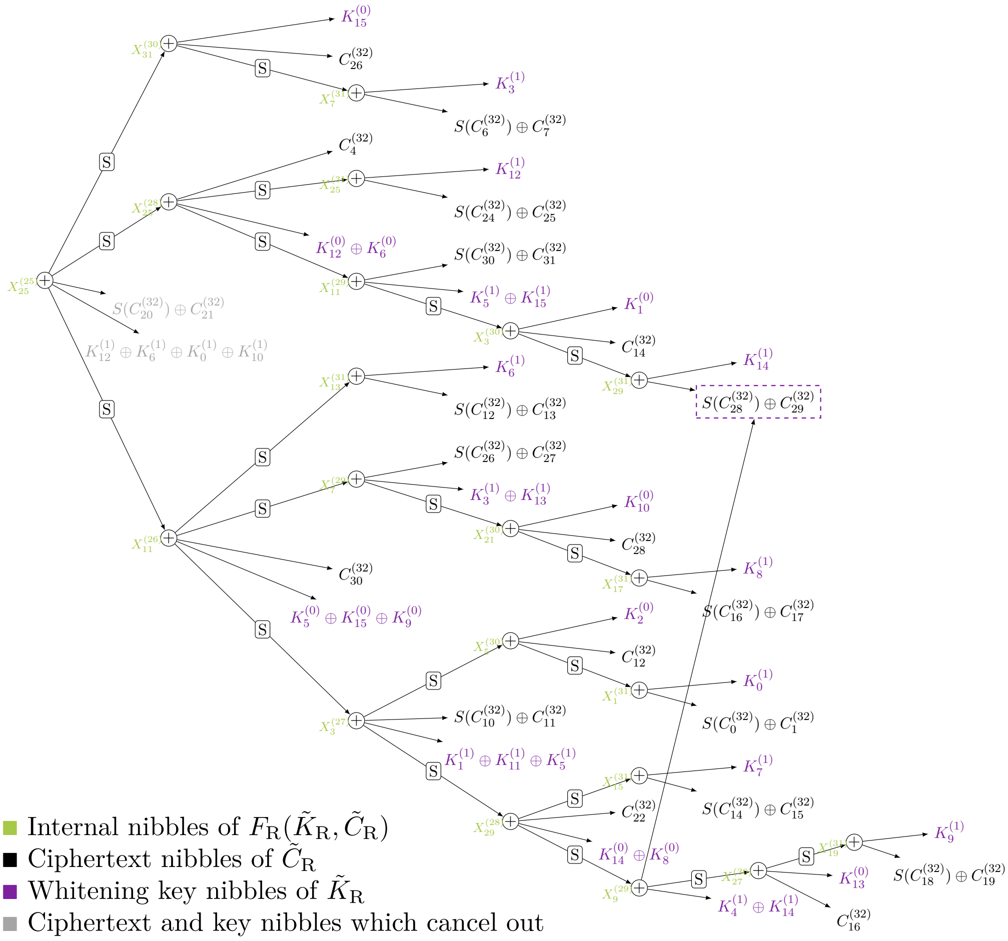
$X_{25}^{(25)}$
$X_{31}^{(30)}$
$X_{25}^{(28)}$
$X_{11}^{(26)}$
$X_{7}^{(31)}$
$X_{25}^{(31)}$
$X_{11}^{(29)}$
$X_{3}^{(30)}$
$X_{29}^{(31)}$
$X_{13}^{(31)}$
$X_{7}^{(29)}$
$X_{21}^{(30)}$
$X_{17}^{(31)}$
$X_{5}^{(30)}$
$X_{1}^{(31)}$
$X_{3}^{(27)}$
$X_{15}^{(31)}$
$X_{29}^{(28)}$
$X_{9}^{(29)}$
$X_{27}^{(30)}$
$X_{19}^{(31)}$
S
$K_{15}^{(0)}$
$C_{26}^{(32)}$
$K_3^{(1)}$
$S(C_6^{(32)}) \oplus C_7^{(32)}$
$C_4^{(32)}$
$K_{12}^{(1)}$
$S(C_{24}^{(32)}) \oplus C_{25}^{(32)}$
$K_{12}^{(0)} \oplus K_6^{(0)}$
$S(C_{30}^{(32)}) \oplus C_{31}^{(32)}$
$K_5^{(1)} \oplus K_{15}^{(1)}$
$K_1^{(0)}$
$C_{14}^{(32)}$
$K_{14}^{(1)}$
$S(C_{28}^{(32)}) \oplus C_{29}^{(32)}$
$S(C_{20}^{(32)}) \oplus C_{21}^{(32)}$
$K_{12}^{(1)} \oplus K_6^{(1)} \oplus K_0^{(1)} \oplus K_{10}^{(1)}$
$K_6^{(1)}$
$S(C_{12}^{(32)}) \oplus C_{13}^{(32)}$
$S(C_{26}^{(32)}) \oplus C_{27}^{(32)}$
$K_3^{(1)} \oplus K_{13}^{(1)}$
$K_{10}^{(0)}$
$C_{28}^{(32)}$
$K_8^{(1)}$
$S(C_{16}^{(32)}) \oplus C_{17}^{(32)}$
$C_{30}^{(32)}$
$K_5^{(0)} \oplus K_{15}^{(0)} \oplus K_9^{(0)}$
$K_2^{(0)}$
$C_{12}^{(32)}$
$K_0^{(1)}$
$S(C_0^{(32)}) \oplus C_1^{(32)}$
$S(C_{10}^{(32)}) \oplus C_{11}^{(32)}$
$K_1^{(1)} \oplus K_{11}^{(1)} \oplus K_5^{(1)}$
$K_7^{(1)}$
$S(C_{14}^{(32)}) \oplus C_{15}^{(32)}$
$C_{22}^{(32)}$
$K_{14}^{(0)} \oplus K_8^{(0)}$
$K_9^{(1)}$
$S(C_{18}^{(32)}) \oplus C_{19}^{(32)}$
$K_{13}^{(0)}$
$K_4^{(1)} \oplus K_{14}^{(1)}$
$C_{16}^{(32)}$
🟩 Internal nibbles of $F_\mathrm{R}(\tilde{K}_\mathrm{R}, \tilde{C}_\mathrm{R})$
■ Ciphertext nibbles of $\tilde{C}_\mathrm{R}$
🟪 Whitening key nibbles of $\tilde{K}_\mathrm{R}$
▨ Ciphertext and key nibbles which cancel out

Figure 5: Dependency DAG for right branch of 9-round MitM key recovery in Figure 4.

Observe that one nibble of key information is irrelevant for recovering the integral sum: The nibble $K_{12}^{(1)} \oplus K_6^{(1)} \oplus K_0^{(1)} \oplus K_{10}^{(1)}$ appears linearly in the sum and will thus cancel out when summed over the total, even number of ciphertexts. The corresponding ciphertext nibble $S(C_{20}^{(32)}) \oplus C_{21}^{(32)}$ sums to a constant that we can evaluate immediately upon collection of the ciphertexts. The remaining 20 equivalent key and 19 ciphertext nibbles appear in paired sums $\tilde{K}_{\mathrm{R}i} \oplus \tilde{C}_{\mathrm{R}i}$, $0 \leq i < 20$, except $S(C_{28}^{(32)}) \oplus C_{29}^{(32)}$ which appears twice with different keys; we can simply include this nibble twice when transforming the ciphertexts C into the compressed ciphertexts $\tilde{C}_\mathrm{R}$ for the right branch X_R. This means that $\tilde{K}_\mathrm{R}$ essentially serves as a whitening key, and the function $F_\mathrm{R}(\tilde{K}_\mathrm{R}, \tilde{C}_\mathrm{R})$ is actually a function $F_\mathrm{R}(\tilde{K}_\mathrm{R} \oplus \tilde{C}_\mathrm{R})$ with input size $k = 80$.

Thus, we can apply the FFT approach as discussed in Subsection 4.2 to each of the 4 bits of X_R. With a complexity of $4 \cdot 4 \cdot k \cdot 2^k = 16 \cdot 80 \cdot 2^{80} = 2^{90.32}$ additions, we obtain a table that maps each of the 2^4 possible values of $\bigoplus_\mathbb{C} X_\mathrm{R} = \bigoplus_\mathbb{C} X_{12}^{(24)}$ to a list of about $2^{80-4} = 2^{76}$ values of the partial key $\tilde{K}_\mathrm{R}$ that produce this sum. By applying an invertible linear function, we can transform the key candidates to the simpler equivalent form

$$\begin{aligned}\tilde{K}_\mathrm{R} = {} & K_1^{(0)}, K_2^{(0)}, K_5^{(0)} \oplus K_9^{(0)}, K_6^{(0)} \oplus K_{12}^{(0)}, K_8^{(0)} \oplus K_{14}^{(0)}, K_{10}^{(0)}, K_{13}^{(0)}, K_{15}^{(0)}, \\ & K_0^{(1)}, K_1^{(1)} \oplus K_5^{(1)} \oplus K_{11}^{(1)}, K_3^{(1)}, K_4^{(1)}, K_5^{(1)} \oplus K_{15}^{(1)}, K_6^{(1)}, \dots K_9^{(1)}, K_{12}^{(1)}, K_{13}^{(1)}, K_{14}^{(1)}.\end{aligned}$$

Left branch: $\bigoplus_{\mathbb{C}} X_{\mathsf{L}} = \bigoplus_{\mathbb{C}} S(X_1^{(24)}) = \bigoplus_{\mathbb{C}} F_{\mathsf{L}}(\tilde{C}_{\mathsf{L}}, \tilde{K}_{\mathsf{L}})$. Similarly to the right branch, we automatically generate the dependency DAG for the left branch with the algorithm of Subsection 4.2. According to the dependency DAG, $X_1^{(24)}$ can be recovered as a function $F_{\mathrm{L}}(\tilde{K}_{\mathrm{L}}, \tilde{C}_{\mathrm{L}})$ using 104 bits (26 distinct nibbles) of partial ciphertext, of which several appear in multiple tree branches, and 128 bits (32 nibbles) of key information. However, these 32 key nibbles only correspond to a space of $2^{28 \cdot 4}$ actual key candidates, as some key terms are linearly dependent. For example, it is easy to see in Figure 4 that $K_{12}^{(0)}, K_{15}^{(0)}$ are not involved in this DAG. Unfortunately, it is not trivial to exploit these dependencies when using the FFT key-recovery approach: For example, if 3 linearly dependent key nodes appear together with 3 distinct ciphertext nodes in completely different parts of the DAG, we still have to first recover the full 3-nibble key material (with the corresponding cost) and can only then filter out inconsistent keys. Furthermore, we cannot handle the multiple shared ciphertext nodes in the same way as in the right branch, as the resulting complexity would be prohibitive.

Postprocessing the DAG. We thus first apply an additional *postprocessing step* to the DAG. Consider again the DAG for the right branch, as illustrated in Figure 5, where one of the `C`-nodes, $S(C_{28}^{(32)}) \oplus C_{29}^{(32)}$, is shared by two parent `X`-nodes. Both parents also share the key nibble $K_{14}^{(1)}$, but due to our merging steps, this is part of two different `K`-nodes. Instead of linking the merged parent $X_9^{(29)}$ only with the `C`-node, we can link it directly with the other `X`-node $X_{29}^{(31)}$ and update its `K`-child from $K_4^{(1)} \oplus K_{14}^{(1)}$ to $K_4^{(1)}$; in other words, we can un-merge the two original Xors combined in $X_9^{(29)}$. The remaining `K`-node $K_4^{(1)}$ now becomes an *internal* key without a corresponding `C`-node. This situation occurs similarly 6 times in the DAG for the left branch, and we update the DAG in the same way, thus creating 6 internal key nodes. In total, we now have 26 ciphertext nodes paired with key nodes, plus the 6 internal key nodes. We refer to the paired key nodes as whitening key nodes. Of the 6 new internal key nodes, 2 involve the same key nibble as a paired whitening key node ($K_6^{(0)}$ and $K_{12}^{(1)}$). The resulting, postprocessed dependency DAG is illustrated in Figure 6.

Key-recovery approach. To take advantage of the linear dependencies between some of the key nodes, we split the key nibbles into three categories (internal keys 🟥, whitening keys 🟪, and internal+whitening keys 🟧 that appear in both roles), and proceed as follows:

- Repeat for each guessed value of the internal+whitening keys 🟧 $K_6^{(0)}$, $K_{12}^{(1)}$ ($2^8\times$):
  1. Transform the nodes depending only on these two keys plus any ciphertext nodes into new synthetic ciphertext nodes 🟧:
     - $X_{25}^{(31)} = K_{12}^{(1)} \oplus S(C_{24}^{(32)}) \oplus C_{25}^{(32)}$, which is now paired with the previous internal key $K_8^{(1)} \oplus K_2^{(1)}$, turning it into a whitening key.
     - $X_{19}^{(29)\prime} = S(S(K_{12}^{(1)} \oplus S(C_{24}^{(32)}) \oplus C_{25}^{(32)}) \oplus C_5^{(32)} \oplus K_6^{(0)}) \oplus S(C_6^{(32)}) \oplus C_7^{(32)}$, which is paired with the previous whitening key $K_9^{(1)} \oplus K_3^{(1)}$.
  2. Repeat for each guessed value of internal keys 🟥 $K_8^{(0)}, K_7^{(1)}, K_{10}^{(1)} \oplus K_4^{(1)}$ ($2^{12}\times$):
     (a) With these 5 internal key bytes fixed, we get a reduced DAG with 25 ciphertext nodes and whitening keys: the 2 new synthetic ones created in the previous step 🟧, plus 23 regular ones ⬛.
     (b) Run the FFT key recovery approach with $k = 4 \cdot 25 = 100$ as before on each of the 4 bits of $X_{\mathrm{L}} = S(X_1^{(24)})$, with a total complexity of $4 \cdot 4 \cdot 100 \cdot 2^{100} = 2^{110.64}$ additions.
     (c) The result is a map from each of the 2^4 values of $\bigoplus_{\mathbb{C}} X_{\mathrm{L}}$ to a list of about $2^{100-4} = 2^{96}$ values of the partial key $\tilde{K}$ that produce this sum. As some of

the involved key nodes are linearly dependent, we can check for consistency and thus reduce this set to $2^{96-4-4} = 2^{88}$ candidates per value:

- Check that $K_8^{(1)} \oplus K_2^{(1)}$, $K_8^{(1)}$, and $K_2^{(1)}$ are consistent
- Check that $K_9^{(1)} \oplus K_3^{(1)} \oplus K_{13}^{(1)} \oplus K_{15}^{(1)}$, $K_9^{(1)} \oplus K_3^{(1)}$, $K_{13}^{(1)} \oplus K_7^{(1)}$, $K_7^{(1)}$, and $K_{15}^{(1)}$ are consistent
- Then, we can apply a linear function to the remaining key candidates to transform them back to the values of the 28 key nibbles $K_0^{(0)}$, $K_1^{(0)} \oplus K_7^{(0)}$, $K_2^{(0)}$, $K_3^{(0)} \oplus K_7^{(0)}$, $K_4^{(0)} \oplus K_{14}^{(0)}$, $K_5^{(0)}$, $K_6^{(0)}$, $K_8^{(0)}$, …, $K_{11}^{(0)}$, $K_{13}^{(0)}$, $K_0^{(1)}$, …, $K_{15}^{(1)}$.

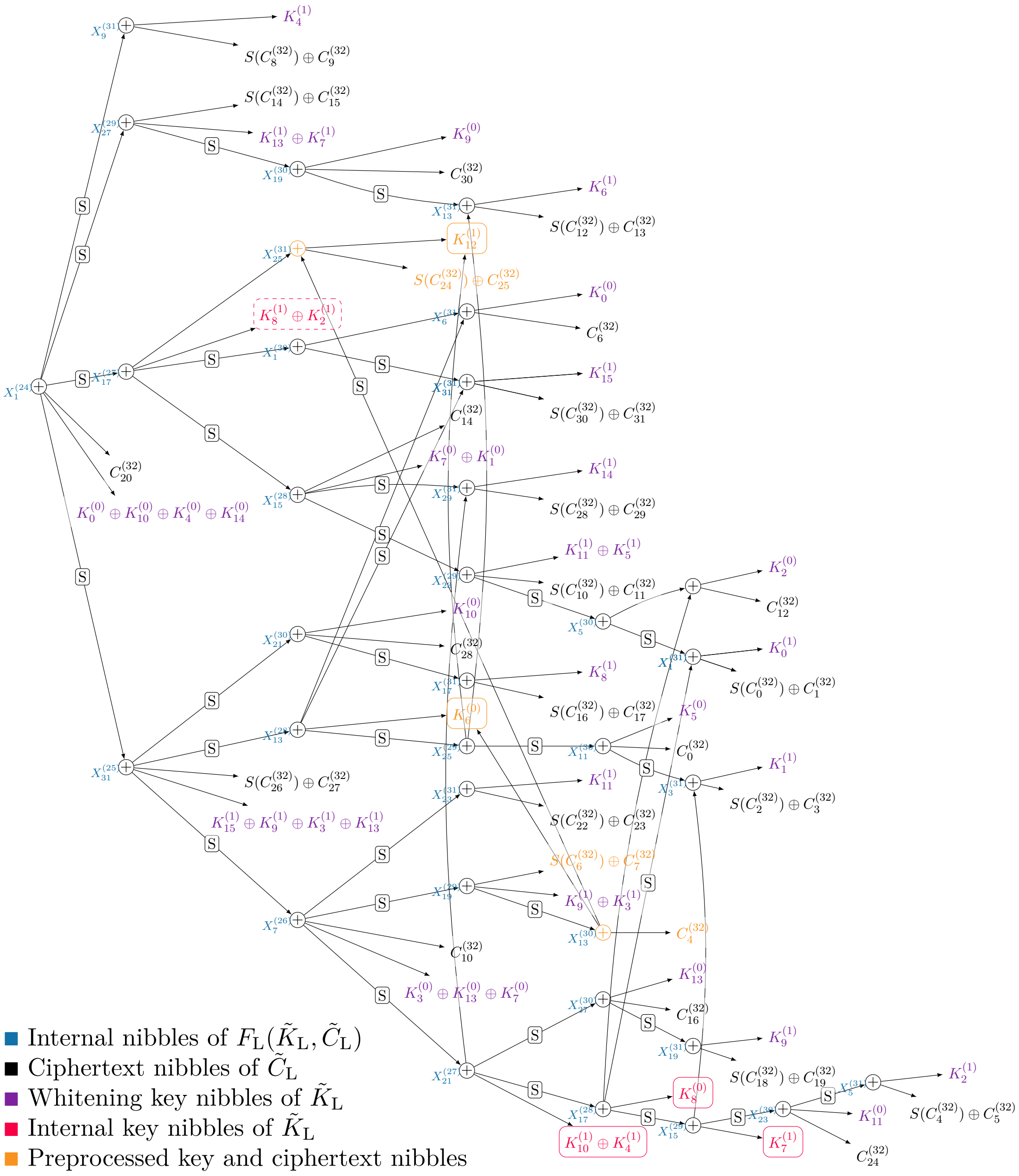
$X_9^{(31)}$
$K_4^{(1)}$
$S(C_8^{(32)}) \oplus C_9^{(32)}$
$S(C_{14}^{(32)}) \oplus C_{15}^{(32)}$
$X_{27}^{(29)}$
$K_{13}^{(1)} \oplus K_7^{(1)}$
$K_9^{(0)}$
$X_{19}^{(30)}$
$C_{30}^{(32)}$
$K_6^{(1)}$
$X_{13}^{(31)}$
$S(C_{12}^{(32)}) \oplus C_{13}^{(32)}$
$X_{25}^{(31)}$
$K_{12}^{(1)}$
$S(C_{24}^{(32)}) \oplus C_{25}^{(32)}$
$K_8^{(1)} \oplus K_2^{(1)}$
$K_0^{(0)}$
$X_6^{(31)}$
$C_6^{(32)}$
$X_1^{(30)}$
$X_1^{(24)}$
$X_{17}^{(25)}$
$K_{15}^{(1)}$
$X_{31}^{(31)}$
$S(C_{30}^{(32)}) \oplus C_{31}^{(32)}$
$C_{14}^{(32)}$
$K_7^{(0)} \oplus K_1^{(0)}$
$C_{20}^{(32)}$
$K_0^{(0)} \oplus K_{10}^{(0)} \oplus K_4^{(0)} \oplus K_{14}^{(0)}$
$X_{15}^{(28)}$
$K_{14}^{(1)}$
$X_{29}^{(31)}$
$S(C_{28}^{(32)}) \oplus C_{29}^{(32)}$
$K_{11}^{(1)} \oplus K_5^{(1)}$
$X_{27}^{(29)}$
$S(C_{10}^{(32)}) \oplus C_{11}^{(32)}$
$K_2^{(0)}$
$C_{12}^{(32)}$
$K_{10}^{(0)}$
$X_5^{(30)}$
$X_{21}^{(30)}$
$C_{28}^{(32)}$
$X_1^{(31)}$
$K_0^{(1)}$
$S(C_0^{(32)}) \oplus C_1^{(32)}$
$K_8^{(1)}$
$X_{17}^{(31)}$
$S(C_{16}^{(32)}) \oplus C_{17}^{(32)}$
$K_5^{(0)}$
$X_{13}^{(29)}$
$K_6^{(0)}$
$X_{25}^{(29)}$
$X_{11}^{(30)}$
$C_0^{(32)}$
$K_1^{(1)}$
$X_{31}^{(25)}$
$X_3^{(31)}$
$S(C_2^{(32)}) \oplus C_3^{(32)}$
$S(C_{26}^{(32)}) \oplus C_{27}^{(32)}$
$X_{23}^{(31)}$
$K_{11}^{(1)}$
$K_{15}^{(1)} \oplus K_9^{(1)} \oplus K_3^{(1)} \oplus K_{13}^{(1)}$
$S(C_{22}^{(32)}) \oplus C_{23}^{(32)}$
$S(C_6^{(32)}) \oplus C_7^{(32)}$
$X_{19}^{(29)}$
$K_9^{(1)} \oplus K_3^{(1)}$
$X_7^{(26)}$
$X_{13}^{(30)}$
$C_4^{(32)}$
$C_{10}^{(32)}$
$K_{13}^{(0)}$
$K_3^{(0)} \oplus K_{13}^{(0)} \oplus K_7^{(0)}$
$X_{27}^{(30)}$
$C_{16}^{(32)}$
$X_{19}^{(31)}$
$K_9^{(1)}$
$X_{21}^{(27)}$
$S(C_{18}^{(32)}) \oplus C_{19}^{(32)}$
$X_5^{(31)}$
$K_2^{(1)}$
$K_8^{(0)}$
$X_{17}^{(28)}$
$X_{23}^{(30)}$
$K_{11}^{(0)}$
$S(C_4^{(32)}) \oplus C_5^{(32)}$
$X_{15}^{(29)}$
$K_{10}^{(1)} \oplus K_4^{(1)}$
$K_7^{(1)}$
$C_{24}^{(32)}$
S
🟦 Internal nibbles of $F_L(\tilde{K}_L, \tilde{C}_L)$
⬛ Ciphertext nibbles of $\tilde{C}_L$
🟪 Whitening key nibbles of $\tilde{K}_L$
🟥 Internal key nibbles of $\tilde{K}_L$
🟧 Preprocessed key and ciphertext nibbles

Figure 6: Dependency DAG for left branch of 9-round MitM key recovery in Figure 4.

As a result, for each of the 2^4 potential values of $\bigoplus_{\mathbb{C}} X_{\mathrm{L}}$ and each of the 2^{20} values of the internal key, we have 2^{88} whitening key candidates $\tilde{K}_{\mathrm{L}}$ that map to this value. With the rough estimate that 1 addition (of less than 128 bits) has the same cost as 1 cipher round or $\frac{1}{32} = 2^{-5}$ cipher evaluations, the total cost of this process is about $2^{8+12} \cdot 2^{110.64} \cdot 2^{-5} = 2^{125.64}$ cipher evaluations and thus less than the data collection phase.

Combining the key information. After the previous step, for each of the 2^4 possible matching values of $\bigoplus_{\mathbb{C}} X_{\mathrm{L}}$ and $\bigoplus_{\mathbb{C}} X_{\mathrm{R}}$, we have 2^{108} key candidates for the 28-nibble key

$$\begin{aligned}\tilde{K}_{\mathrm{L}} = {} & K_0^{(0)}, K_1^{(0)} \oplus K_7^{(0)}, K_2^{(0)}, K_3^{(0)} \oplus K_7^{(0)}, K_4^{(0)} \oplus K_{14}^{(0)}, K_5^{(0)}, K_6^{(0)}, K_8^{(0)}, \ldots, K_{11}^{(0)}, K_{13}^{(0)}, \\ & K_0^{(1)}, \ldots, K_{15}^{(1)}\end{aligned}$$

and 2^{76} candidates for the 21-nibble key

$$\begin{aligned}\tilde{K}_{\mathrm{R}} = {} & K_1^{(0)}, K_2^{(0)}, K_5^{(0)} \oplus K_9^{(0)}, K_6^{(0)} \oplus K_{12}^{(0)}, K_8^{(0)} \oplus K_{14}^{(0)}, K_{10}^{(0)}, K_{13}^{(0)}, K_{15}^{(0)}, \\ & K_0^{(1)}, K_1^{(1)} \oplus K_5^{(1)} \oplus K_{11}^{(1)}, K_3^{(1)}, K_4^{(1)}, K_5^{(1)} \oplus K_{15}^{(1)}, K_6^{(1)}, \ldots K_9^{(1)}, K_{12}^{(1)}, K_{13}^{(1)}, K_{14}^{(1)}.\end{aligned}$$

We can match them efficiently by sorting the values for $\tilde{K}_{\mathrm{L}}$ and $\tilde{K}_{\mathrm{R}}$ based on the 12 nibbles of $K^{(1)}$ involved in $\tilde{K}_{\mathrm{R}}$ and the 4 nibbles $K_2^{(0)}$, $K_{10}^{(0)}$, $K_{13}^{(0)}$, $K_5^{(0)} \oplus K_9^{(0)}$, leaving about $2^{76-4\cdot16} = 2^{12}$ candidates for $\tilde{K}_{\mathrm{R}}$ and $2^{108-4\cdot16} = 2^{44}$ candidates for $\tilde{K}_{\mathrm{L}}$ per matching value. Then, for each of the $2^4 \cdot 2^{4\cdot16} = 2^{68}$ values of the combined matching value (target sum X and partial key), we get $2^{12} \cdot 2^{44} = 2^{56}$ candidates for the complete key, as the rest of $\tilde{K}_{\mathrm{L}}$ and $\tilde{K}_{\mathrm{R}}$ together fully determines the key. The resulting $2^{68} \cdot 2^{56} = 2^{124}$ key candidates can be tested with brute-force search.

Complexity. Overall, the time and data complexity is dominated by the data requirement of 2^{127} chosen-plaintext queries for the distinguisher. When using a shorter distinguisher to achieve a less marginal complexity, the dominating complexity is 2^{124} for exhaustively testing the remaining key candidates. The memory complexity depends on the required memory for FWHT with $k = 100$ using about 2^{100} counters, plus storing the corresponding partial ciphertexts using $2^{26\cdot4} = 2^{104}$ bits, plus 2^{108} partial key candidates (all for the left branch in our 32-round integral attack), so in total less than 2^{108} 128-bit states with some optimization potential by rearranging the key-combining step.

5 Conclusion

`WARP` achieves its design goals with the help of a generalized Feistel network with a high number of small branches as well as some special design choices like a low-area S-box and a key addition after this S-box in the Feistel-F-function. Since the focus is on low area, the function has a relatively slow diffusion. Previous analysis results only take these properties into account to a limited extent. For example, the previous 21-round integral attack sketched by the designers takes neither the bitwise S-box definition nor the key position into account for potential improvements.

We show how a more detailed analysis of these properties, such as the ANF and bitwise Monomial Prediction properties of the S-box, allow us to target significantly more rounds with a comparable complexity. For our 32-round attack, we used our implementation of Hu et al.'s monomial prediction as well as manual observations to extend the distinguisher by 4 rounds. To cover as many rounds as possible in the key-recovery phase despite the high data complexity of the underlying distinguisher, we implemented a graph-based model of the intermediate variables required for key recovery as a directed acyclic graph. We showed how to restructure this graph in order to obtain a representation that works well with optimized key-recovery techniques, particularly FFT key recovery. Our improvements provide a tighter estimate of the security margin of `WARP`, but do not threaten its security.

References

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric Even-Mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017. doi:10.13154/tosc.v2017.i1.4-44.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015. doi:10.1007/978-3-662-48800-3_17.

[BBI+20] Subhadeep Banik, Zhenzhen Bao, Takanori Isobe, Hiroyasu Kubo, Fukang Liu, Kazuhiko Minematsu, Kosei Sakamoto, Nao Shibata, and Maki Shigeri. WARP: Revisiting GFN for lightweight 128-bit block cipher. In *SAC 2020*, volume 12804 of *LNCS*, pages 535–564. Springer, 2020. doi:10.1007/978-3-030-81652-0_21.

[BHMSV84] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2 of *The Kluwer International Series in Engineering and Computer Science*. Springer, 1984.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007. doi:10.1007/978-3-540-74735-2_31.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present – towards reaching the limit of lightweight encryption. In *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017. doi:10.1007/978-3-319-66787-4_16.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In *FSE 1997*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997. doi:10.1007/BFb0052343.

[EKKT18] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding integral distinguishers with ease. In *SAC 2018*, volume 11349 of *LNCS*, pages 115–138. Springer, 2018. doi:10.1007/978-3-030-10970-7_6.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In *FSE 2000*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000. doi:10.1007/3-540-44706-7_15.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011. doi:10.1007/978-3-642-23951-9_22.

[GPV19] Kishan Chand Gupta, Sumit Kumar Pandey, and Ayineedi Venkateswarlu. Almost involutory recursive MDS diffusion layers. *Des. Codes Cryptogr.*, 87(2-3):609–626, 2019. doi:10.1007/s10623-018-0582-2.

[HSWW20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In *ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020. doi:10.1007/978-3-030-64837-4_15.

[KW02] Lars R. Knudsen and David A. Wagner. Integral cryptanalysis. In *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002. doi:10.1007/3-540-45661-9_9.

[KY21a] Manoj Kumar and Tarun Yadav. MILP based differential attack on round reduced WARP. In *SPACE 2021*, volume 13162 of *LNCS*, pages 42–59. Springer, 2021. doi:10.1007/978-3-030-95085-9_3.

[KY21b] Manoj Kumar and Tarun Yadav. MILP based differential attack on round reduced WARP. In *SPACE 2021*, volume 13162 of *LNCS*, pages 42–59. Springer, 2021. doi:10.1007/978-3-030-95085-9_3.

[Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. In *Communications and cryptography*, pages 227–233. Springer, 1994.

[McC56] Edward J McCluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[Qui52] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952.

[SIH^{+}11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 342–357. Springer, 2011. doi:10.1007/978-3-642-23951-9_23.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects – revealing structural properties of several ciphers. In *EUROCRYPT 2017*, volume 10212 of *LNCS*, pages 185–215. Springer, 2017. doi:10.1007/978-3-319-56617-7_7.

[SW12] Yu Sasaki and Lei Wang. Meet-in-the-middle technique for integral attacks against Feistel ciphers. In *SAC 2012*, volume 7707 of *LNCS*, pages 234–251. Springer, 2012. doi:10.1007/978-3-642-35999-6_16.

[TA14] Yosuke Todo and Kazumaro Aoki. FFT key recovery for integral attack. In *CANS 2014*, volume 8813 of *LNCS*, pages 64–81. Springer, 2014. doi:10.1007/978-3-319-12280-9_5.

[TB21] Je Sen Teh and Alex Biryukov. Differential cryptanalysis of WARP. Cryptology ePrint Archive, Report 2021/1641, 2021. URL: https://ia.cr/2021/1641.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. In *ACNS 2011*, volume 6715 of *LNCS*, pages 327–344, 2011. doi:10.1007/978-3-642-21554-4_19.