Cryptanalysis of QARMAv2

Anonymous Submission

Abstract. QARMAv2 is a general-purpose and hardware-oriented family of lightweight tweakable block ciphers (TBCs) introduced in ToSC 2023. QARMAv2, as a redesign of QARMA with a longer tweak and tighter security margins, is also designed to be suitable for cryptographic memory protection and control flow integrity. The designers of QARMAv2 provided a relatively comprehensive security analysis in the design specification, e.g., some bounds for the number of attacked rounds in differential and boomerang analysis, together with some concrete impossible differential, zero-correlation, and integral distinguishers. As one of the first third-party cryptanalysis of QARMAv2, Hadipour et al., [HGSE23] significantly improved the integral distinguishers of QARMAv2, and provided the longest concrete distinguishers of QARMAv2 up to now. However, they provided no key recovery attack based on their distinguishers. This paper delves into the cryptanalysis of QARMAv2 to enhance our understanding of its security. Given that the integral distinguishers of QARMAv2 are the longest concrete distinguishers for this cipher so far, we focus on integral attack. To this end, we first further improve the automatic tool introduced by Hadipour et al. [HSE23, HGSE23] for finding integral distinguishers of TBCs following the TWEAKEY framework. This new tool exploits the MixColumns property of QARMAv2 to find integral distinguishers more suitable for key recovery attacks. Then, we combine several techniques for integral key recovery attacks, e.g., Meet-in-the-middle and partial-sum techniques to build a fine-grained integral key recovery attack on QARMAv2. Notably, we demonstrate how to leverage the low data complexity of the integral distinguishers of QARMAv2 to reduce the memory complexity of the meet-in-the-middle technique. As a result, we managed to propose the first concrete key recovery attacks on reduced-round versions of QARMAv2, by attacking 13 rounds of QARMAv2-64-128 with a single tweak block, 14 rounds of QARMAv2-64-128 with two independent tweak blocks, and 16 rounds of QARMAv2-128-256 with two independent tweak blocks. Our attacks do not compromise the claimed security of QARMAv2, but they shed more light on the cryptanalysis of this cipher.

Keywords: Cryptanalysis · Integral attacks · Partial-sum technique · Constraint programming · QARMAv2

1 Introduction

Our computing devices perform a wide range of computations, some of which are very sensitive, e.g., cryptographic operations, and others might even be malicious, e.g., malware. In addition, with the growth of cloud computing, we increasingly rely on running mutually untrusted processes on a shared platform. Overall, the adversary may have access to the system in use and can run a task on the same platform as the victim. In this security model, we trust the host hardware/platform but not the software running on it. Therefore, it's crucial to safeguard the sensitive parts of codes and data from unauthorized access by other processes on the same platform, ensuring that these operations and data remain private and secure. One solution is cryptographic memory protection to guarantee confidentiality and control flow integrity [Com16, Sec17].

One example is the Pointer Authentication Code (PAC) [Com16, Sec17], used in Arm architectures, which provides a control flow integrity mechanism and makes it much harder

for an attacker to modify protected pointers in memory without being detected. The idea behind PAC is to insert a PAC into each pointer we want to protect before writing it to memory, and then verify the PAC before using the pointer. Therefore, an adversary who aims to modify a protected pointer has to find the correct PAC for the new value of the pointer to control the program flow. Another example of cryptographic memory protection can be found in Intel's SGX technology (Software Guard Extensions) [Gue16] incorporated into their CPUs. SGX creates a secure enclave within the processor, providing a protected area where sensitive operations, including encryption tasks, can occur without being accessible to external interference. The enclave ensures the confidentiality and integrity of the data and code within it, offering a secure execution environment even when the broader system may not be fully trusted.

The cryptographic primitives required for memory encryption, should be very fast to minimize the performance overhead. At the same time, they should be secure enough. Therefore, latency is the primary engineering constraint in the design of lightweight block ciphers for memory encryption, whereas area and, thus, the power are the secondary constraints. The previously well-analyzed ciphers, such as AES, are not a good choice because their latency is too high for memory encryption. In addition, for efficient memory encryption, we need a cryptographic primitive where the permutation not only depends on the key and plaintext but also on a public parameter *tweak* that can be the encrypted block's physical address. One approach to achieve this is to use modes of operations based on classical block ciphers. But these modes typically require constructions that lead to increased latency or extra memory to store, for example, the nonce. Another approach is to use a *tweakable block cipher* (TBC), where the permutation is determined by the secret parameter key and public parameters tweak and plaintext. In a TBC, the cipher should remain secure even if the tweak can be controlled by the adversary.

`QARMAv2` [ABD+23] is a general-purpose and hardware-oriented family of lightweight TBC that is designed to be also suitable for cryptographic memory protection and control flow integrity. This paper explores `QARMAv2` from the cryptanalysis aspect, shedding light on its security against cryptanalytic attacks. The designers of `QARMAv2` provided a relatively comprehensive security analysis in the design specification, e.g., differential, boomerang, integral, impossible differential, and zero-correlation attacks. For instance, the designers used the method introduced initially in [HBS21, HNE22] to provide some *bounds* for the number of attacked rounds in boomerang analysis. They also used the methods introduced very recently at EUROCRYPT 2023 [HSE23] to provide some concrete impossible differential and zero-correlation distinguishers. As another example, they used division property [Tod15, XZBL16] to provide concrete integral distinguishers for up to 5 rounds of `QARMAv2`-64.

As a first third party cryptanalysis of `QARMAv2`, Tim Beyne [Bey23] found a nonlinear invariant for the unkeyed round function of `QARMAv2`-64, a property that can be extended to multiple rounds only for a set of weak keys, but does not affect the full-round `QARMAv2`-64. Also, the designers addressed this weakness by incorporating a new S-box in the final version of the QARMAv2 specification [ABD+23]. As another third-party analysis, very recently, Hadipour et al., [HGSE23] significantly improved the integral distinguishers of `QARMAv2`. The longest *concrete* distinguishers for `QARMAv2` up to now are the integral distinguishers proposed in [HGSE23]. The authors of [HGSE23] exploited the control of the adversary over the tweak part and improved the automatic tool introduced in [HSE23] to find integral distinguishers for up to 10 (resp. 12) rounds of `QARMAv2`-64 (resp. `QARMAv2`-128). However, they did not provide any key recovery attack based on their distinguishers, and the efficiency of integral key recovery attacks for `QARMAv2` is still an open question. Therefore, this paper focuses on the integral cryptanalysis of `QARMAv2`.

Table 1: Summary of our attacks on QARMAv2. $\mathscr{T}$: No. of independent tweak blocks.

Version	$\mathscr{T}$	#Rounds	Time	Data	Memory	Ref.
QARMAv2-64-128	1	13/16	$2^{110.47}$	$2^{46.32}$	$2^{46.32}$	Subsection 5.1
QARMAv2-64-128	2	14/20	$2^{110.17}$	$2^{46.32}$	$2^{46.32}$	Subsection 5.2
QARMAv2-128-256	2	16/32	$2^{234.11}$	$2^{46.58}$	$2^{52.00}$	Subsection 5.3

Our contributions. In this paper, we shed more light on the security of QARMAv2. Considering that the integral distinguishers of QARMAv2 are the longest concrete distinguishers for this cipher so far, we focus on integral attack. We first improve the automatic tool introduced in [HSE23, HGSE23] for finding integral distinguishers of TBCs following the TWEAKEY framework. This new tool exploits the MixColumns property of QARMAv2 to find integral distinguishers more suitable for key recovery attacks. The application of our new CP model to find integral distinguishers is not limited to QARMAv2, and it is applicable to other TBCs such as MANTIS and CRAFT [BLMR19]. Then, we combine several techniques for integral key recovery attacks, e.g., Meet-in-the-middle [SW12] and partial-sum [FKL+00] techniques to build a fine-grained integral key recovery attack on QARMAv2. Notably, we demonstrate how to leverage the low data complexity of the integral distinguishers of QARMAv2 to reduce the memory complexity of the meet-in-the-middle technique. Table 1 summarizes our key recovery attacks. While the designers of QARMAv2 assert $(\kappa - \varepsilon)$-bit security for a κ bit secret key, with ε as a small number, such as 2, they recommend larger values for ε, such as 16 for standardization. Our analyses remain valid even for standardization purposes, i.e., $\varepsilon \leq 16$. However, our attacks do not compromise the claimed security of QARMAv2. Additionally, our attacks operate in the unbalanced setting, meaning that the number of forward and backward rounds is not equal.

Outline. We first recall the specification of QARMAv2 in Subsection 2.1. It is followed by a brief overview of the integral distinguishers and their relation to zero-correlation distinguishers in Subsection 2.2. Subsection 2.3 briefly reviews the partial-sum and meet-in-the-middle techniques in the key recovery of integral attacks. Section 3 discusses the MixColumns property of QARMAv2 in terms of integral cryptanalysis. After that, we present our improved automatic tool for finding integral distinguishers of TBCs following the TWEAKEY framework in Section 4. Lastly, we present our integral key recovery attacks on QARMAv2 in Section 5 and conclude in Section 6.

2 Background

In this section, we review the QARMAv2 specification. We then provide a brief overview of ZC distinguishers and their conversion to integral distinguishers for block ciphers. Lastly, we cover the partial-sum and meet-in-the-middle techniques in the key recovery of integral attacks.

2.1 Specification of QARMAv2

QARMAv2 is a redesign of QARMA with a longer tweak and tighter security margins that was introduced in ToSC 2023 [ABD+23]. The goal behind the design of QARMAv2 is to provide a general TBC that is also suitable for memory encryption, and fast computation of short-message MACs. It offers two block sizes, $b = 64$, 128 bits, denoted by QARMAv2-b-s, where s is the bit size of the key (or the security level in bits). For $b = 128$, the key size can be $s = 128$, 192, or 256 bits, and for $b = 64$, the key length is always $s = 128$ bits and can be omitted from the notation. Similar to MANTIS [BJK+16], and PRINCE [BCG+],

QARMAv2 also follows the *reflector construction* as illustrated in Figure 1. Following the specification of QARMAv2 [ABD+23], we represent the inverse of a function f by $\overline{f}$, and f^{-1} interchangeably, in this paper.

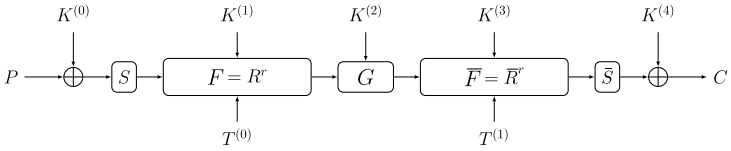


Figure 1: Reflector structure of QARMAv2

As Figure 1 shows, the reflector construction consists of three parts: forward function F, backward function $\overline{F} = F^{-1}$, and the central construction G. This construction allows the implementation of both encryption and decryption using the same circuit with a minor set-up cost. According to Figure 1, the first and the last rounds include only the S-box layer and key addition without mixing with a tweak. The reflector is also independent of a tweak. $K^{(i)}$, (resp. $T^{(i)}$) are derived by applying a linear function on a master key K (resp. master tweak T).

Algorithm 1: The QARMAv2 algorithm.

Input: op, r, K_0, K_1, W_0, W_1, T_0, T_1, P
Output: C

$t_0 \leftarrow T_0,\ t_1 \leftarrow T_1$ /* Round tweak setup */
$k_0 \leftarrow K_0,\ k_1 \leftarrow K_1$ /* Round key setup */
$X \leftarrow S\left(P \oplus k_0\right)$ /* Round #0 */
for $i = 1, \ldots, r$ **do**
  /* Rounds #1 to #r */
  $X \leftarrow X \oplus k_{i \bmod 2} \oplus t_{i \bmod 2} \oplus c_i$;
  $X \leftarrow \left(S \circ M \circ \tau\right)(X)$;
  if $i \equiv 1 \mod 2$ **then** $t_1 \leftarrow \varphi(t_1)$ **else** $t_0 \leftarrow \overline{\varphi}(t_0)$;
  {**if** $i \equiv r \mod 2$ **then** $X \leftarrow$ eXchangeRows (X)} /* Only for $\ell = 2$ */
$k_0 \leftarrow o\left(k_0\right),\ k_1 \leftarrow \overline{o}\left(k_1\right)$;
if *op* = *enc* **then**
  $k_0 \leftarrow k_0 \oplus \alpha,\ k_1 \leftarrow k_1 \oplus \beta$
else
  $k_0 \leftarrow k_0 \oplus o\left(\beta\right),\ k_1 \leftarrow k_1 \oplus o^{-1}\left(\alpha\right)$
$X \leftarrow \overline{\tau}\left(M \cdot \left(\tau\left(X\right) \oplus W_{r+1 \bmod 2}\right) \oplus W_{r \bmod 2}\right)$ /* Reflector */
for $i = r, \ldots, 1$ **do**
  /* Rounds #$r+1$ to #$2r$ */
  {**if** $i \equiv r \mod 2$ **then** $X \leftarrow$ eXchangeRows (X)}/* Only for $\ell = 2$ */
  $X \leftarrow \left(\overline{\tau} \circ \overline{M} \circ \overline{S}\right)(X)$;
  $X \leftarrow X \oplus k_{i+1 \bmod 2} \oplus t_{i+1 \bmod 2} \oplus c_i$;
  if $i > 1$ **and** $i \equiv 0 \mod 2$ **then** $t_1 \leftarrow \varphi\left(t_1\right)$ **else** $t_0 \leftarrow \overline{\varphi}\left(t_0\right)$;
$C \leftarrow \overline{S}\left(X\right) \oplus k_1$ /* Round #$2r+1$ */
return $\mathcal{M}$;

The algorithm 1 describes the encryption/decryption of QARMAv2 in detail. X in algorithm 1 represents the internal state of the cipher and can be considered as ℓ layers of 4×4 arrays of nibbles, where $\ell \in \{1, 2\}$. The data is arranged row-wise in each layer as follows:

$$X = x_0||x_1||\cdots||x_{15} = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}$$

In addition, assuming that b is the block size of QARMAv2-b, the first cell of the first layer includes bit indices $[b-1, \cdots, b-4]$, and the last cell of the last layer includes bit

indices $[3, \cdots, 0]$. Consequently, the number of layers is $\ell = b/64$. Besides, a b-bit value in the design of `QARMAv2` is called a *block*. In what follows, we briefly describe the operations in `QARMAv2` encryption in algorithm 1. The round constraints c_i in algorithm 1 have no impact on our analysis and we omit them.

The state shuffle τ is applied to each layer separately, rearranging the nibbles' positions as shown in Figure 2.

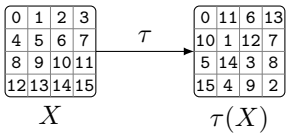


Figure 2: State shuffle τ of `QARMAv2`.

The S-box layer, denoted as S, applies a 4-bit S-box to each nibble of the state. `QARMAv2` employs the following S-box for general-purpose applications:

$$S = [\texttt{4}, \texttt{7}, \texttt{9}, \texttt{b}, \texttt{c}, \texttt{6}, \texttt{e}, \texttt{f}, \texttt{0}, \texttt{5}, \texttt{1}, \texttt{d}, \texttt{8}, \texttt{3}, \texttt{2}, \texttt{a}], \tag{1}$$

and offers the following S-box for Pointer Authentication Code (PAC), or memory authentication use cases:

$$\sigma_0 = [\texttt{0}, \texttt{e}, \texttt{2}, \texttt{a}, \texttt{9}, \texttt{f}, \texttt{8}, \texttt{b}, \texttt{6}, \texttt{4}, \texttt{3}, \texttt{7}, \texttt{d}, \texttt{c}, \texttt{1}, \texttt{5}]. \tag{2}$$

The MixColumns layer M multiplies the following matrix to each column of each layer:

$$M \coloneqq \mathrm{circ}\left(0, \rho, \rho^2, \rho^3\right) = \begin{pmatrix} 0 & \rho & \rho^2 & \rho^3 \\ \rho^3 & 0 & \rho & \rho^2 \\ \rho^2 & \rho^3 & 0 & \rho \\ \rho & \rho^2 & \rho^3 & 0 \end{pmatrix}, \tag{3}$$

where $\rho \in \mathbb{F}_2^4$, and $\rho^4 = 1$. In other words, ρ is the rotation to the left by one bit, i.e., $\rho\left((x_3, x_2, x_1, x_0)\right) = (x_2, x_1, x_0, x_3)$, for $x = (x_3, x_2, x_1, x_0) \in \mathbb{F}_2^4$.

The `exChangeRows` operation is exclusively employed in the case of 2-layer versions ($\ell = 2$). It involves swapping the first two rows between the two layers. This operation is applied every second round in forward and backward rounds, and should always appear in rounds r, and $r + 1$, i.e., before and after the central construction.

The tweakey schedule of `QARMAv2` closely follows the `TWEAKEY` framework [JNP14], but it distinguishes the key and tweak, maintaining separate schedules for each. For $\ell = 1$ (i.e., QARMAv2-64), the only acceptable key size is 128. For $\ell = 2$ (i.e., QARMAv2-128), encryption is always defined with two full 128-bit inputs for the key, i.e., for a 256-bit string $K_0||K_1$. In the case of QARMAv2-128-256, K_0 and K_1 are two halves of the master key. For other versions of QARMAv2-128 with a master key shorter than 256 bits, the master key is first extended to a 256-bit extended key $K_0||K_1$. For more details about the extension function, refer to [ABD+23]. Then, the encryption algorithm alternates between using K_0 and K_1 as the round keys for the forward rounds. For the backward rounds it uses L_0, and L_1 as the round keys which are derived from K_0, and K_1, by the following linear transformations:

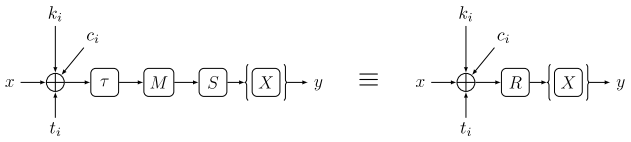
$$(L_0, L_1) \coloneqq \left(o(K_0) \oplus \alpha, o^{-1}(K_1) \oplus \beta\right), \tag{4}$$

where α, and β are constants and o is a linear function over $\mathbb{F}_2^b$ as follows: $o(w) \coloneqq (w \ggg 1) \oplus (w \gg (b-1))$. For the reflector part, it uses $W_0 = o^2(K_0)$, and $W_1 = o^{-2}(K_1)$, as the round keys (see Figure 4b).

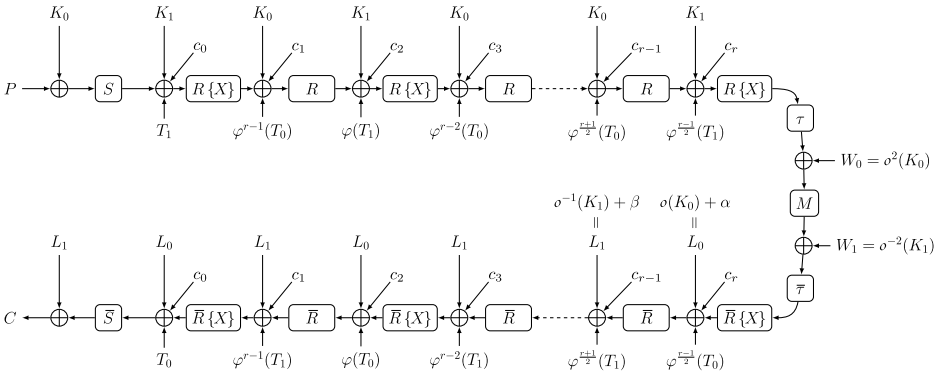
Let T denote the master tweak. Also, let $\mathcal{T}$ represent the number of independent tweak blocks. For $\mathcal{T} = 1$, we define $T_1 = \varphi(T_0)$, where $T_0 = T$. Besides, the two tweak blocks just before the center for encryption are equal in the case of $\mathcal{T} = 1$. For $\mathcal{T} = 2$, T_0, and T_1 are two independent blocks (each one b bits) of the master tweak $T = T_0||T_1$. Let t_i be the round tweak in round i. Besides, assume that $t_1 = T_1$, and $t_2 = \varphi^{-r}(T_0)$, where r is the number of forward/backward rounds. The tweak schedule of QARMAv2 derives the round tweaks as follows: $t_{2i+1} = \varphi(t_{2i-1})$, and $t_{2i+2} = \varphi^{-1}(t_{2i})$ for $i \geq 1$, where φ is a permutation on the position of the tweak nibbles as illustrated in Figure 3.

(a) Tweak shuffle φ of QARMAv2 ($\ell = 1$). (b) Tweak shuffle φ of QARMAv2 ($\ell = 2$).

Figure 3: Tweak shuffle of QARMAv2.



(a) A full round of QARMAv2.



(b) QARMAv2 encryption for odd r.

Figure 4: Overall view of QARMAv2 encryption.

One round of QARMAv2 is represented in Figure 4a, and in Figure 4b, you can see the QARMAv2 encryption for odd values of r, where the number of forward and backward rounds are the same. Table 2 briefly describes the main parameters of different versions of QARMAv2. As per [ABD+23], ε in Table 2 is typically a small number like 2. However, for standardization, the QARMAv2 designers recommend setting ε to 16.

In the context of cryptanalysis, we need to study the security of reduced round versions of QARMAv2. A reduced round version of QARMAv2 can be obtained by reducing the number of rounds before or after the reflector construction, or both. According to the designers in [ABD+23], rounds are counted as S-box layers. If the number of rounds before and after the reflector construction are the same, we call it a *balanced* reduced round of QARMAv2,

otherwise it is called an *unbalanced* reduced round. We recall that, the designers of QARMAv2 also provided cryptanalysis results in unbalanced setting in [ABD+23]. For example, the impossible-differential distinguisher for 9 rounds of QARMAv2-64 ($\mathscr{T} = 2$) in [ABD+23] is composed of 5 forward rounds and 4 backward rounds. As another example, most of the *bounds* for boomerang distinguishers in [ABD+23], apply to unbalanced reduced rounds.

Table 2: Main parameters of QARMAv2

(a) Parameters of QARMAv2 with two tweak blocks ($\mathscr{T} = 2$).

Version	Block size (b)	Key Size (s)	r	#Rounds	Time	Data
QARMAv2-64-128	64	128	9	20	$2^{128-\varepsilon}$	2^{56}
QARMAv2-128-128	128	128	11	24	$2^{128-\varepsilon}$	2^{80}
QARMAv2-128-192	128	192	13	28	$2^{192-\varepsilon}$	2^{80}
QARMAv2-128-256	128	256	15	32	$2^{256-\varepsilon}$	2^{80}

(b) Parameters of QARMAv2 with a single tweak block ($\mathscr{T} = 1$).

Version	Block size (b)	Key Size (s)	r	#Rounds	Time	Data
QARMAv2-64-128	64	128	7	16	$2^{128-\varepsilon}$	2^{56}
QARMAv2-128-128	128	128	9	20	$2^{128-\varepsilon}$	2^{80}
QARMAv2-128-192	128	192	11	24	$2^{192-\varepsilon}$	2^{80}
QARMAv2-128-256	128	256	13	28	$2^{256-\varepsilon}$	2^{80}

2.2 From Zero-Correlation to Integral Distinguishers

The concept of integral distinguishers was initially introduced as a theoretical extension of differential distinguishers by Lai [Lai94] and subsequently, as a practical attack by Daemen et al., [DKR97]. This concept was further formalized by Knudsen and Wagner [KW02]. The fundamental idea behind integral distinguishers is to identify a set of inputs whose corresponding outputs sum up to zero (or a key-independent value) in specific bit/cell positions. The idea of zero-correlation (ZC) distinguishers was initially proposed by Bogdanov and Rijmen [BR14] after introducing integral distinguishers. The core idea of ZC distinguishers is to exploit the linear approximations with zero correlation of the block cipher to distinguish it from a random permutation. ZC attacks were later improved further by Bogdanov and Wang at FSE 2012 in [BW12].

At ASIACRYPT 2012, Bogdanov et al. demonstrated in [BLNW12] that an integral distinguisher, as defined by a balanced vectorial Boolean function, unconditionally implies a ZC distinguisher. At CRYPTO 2015, Sun et al. introduced Theorem 1 in [SLR+15] which states that a ZC linear hull for block ciphers defined over $\mathbb{F}_2^n$ always results in an integral distinguisher.

Theorem 1 (Sun et al. [SLR+15])**.** *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a vectorial Boolean function. Assume A is a subspace of $\mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^n \setminus \{0\}$ such that (α, β) is a ZC approximation for any $\alpha \in A$. Then, for any $\lambda \in \mathbb{F}_2^n$, $\langle \beta, F(x + \lambda)\rangle$ is balanced over the set*

$$A^{\perp} = \{x \in \mathbb{F}_2^n \mid \forall\ \alpha \in A : \langle \alpha, x\rangle = 0\}.$$

According to Theorem 1, the data complexity of the integral distinguisher obtained from a ZC linear hull is 2^{n-m}, where n is the block size, and m is the dimension of the

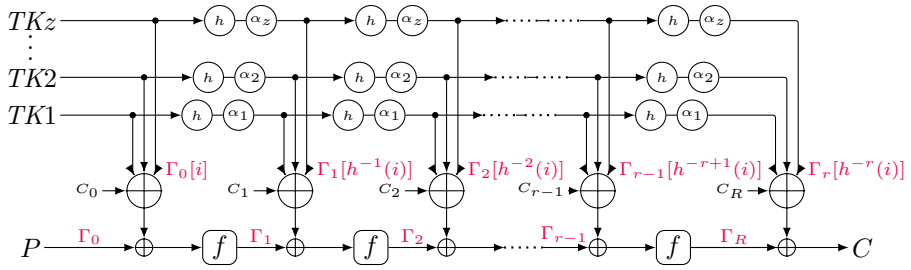


Figure 5: The STK construction of the TWEAKEY framework.

linear space created by the input linear masks in the corresponding ZC linear hull. At ToSC 2019, Ankele et al. investigated the impact of the tweakey on ZC distinguishers for TBCs, as discussed in [ADG+19]. Their research revealed that considering the tweakey schedule can lead to longer ZC and integral distinguishers. They introduced Theorem 2, offering an algorithmic approach to find ZC linear hulls for TBCs based on the super-position tweakey (STK) construction within the TWEAKEY framework [JNP14].

Theorem 2 (Ankele et al. [ADG+19]). *Let $E_K(T, P) : \mathbb{F}_2^{t\times n} \to \mathbb{F}_2^n$ be a TBC following the STK construction as illustrated in Figure 5. Suppose that the tweakey schedule of E_K has z parallel paths and applies a permutation h on the tweakey cells in each path. Let (Γ_0, Γ_r) be a pair of linear masks for r rounds of E_K, and $\Gamma_1, \ldots, \Gamma_{r-1}$ represents a possible sequence for the intermediate linear masks. If there is a cell position i such that any possible sequence $\Gamma_0[i], \Gamma_1[h^{-1}(i)], \Gamma_2[h^{-2}(i)], \ldots \Gamma_r[h^{-r}(i)]$ has at most z linearly active cells, then (Γ_0, Γ_r) yields a ZC linear hull for r rounds of E.*

At EUROCRYPT 2023, Hadipour et al. introduced a new CP/MILP modeling [HSE23], to find ZC linear hulls for TBCs based on Theorem 2, and significantly enhanced the ZC and integral attacks on all variants of SKINNY and some other tweakable block ciphers. This CP model was further improved in [HGSE23].

The QARMAv2 design is related to TWEAKEY framework. Besides, the methods introduced in [HSE23, HGSE23] have proven highly efficient in uncovering integral distinguishers for TBCs using the TWEAKEY construction. Consequently, we employ the same approach to discover integral distinguishers for QARMAv2. Nevertheless, as we will elaborate in Section 4, we refine the technique introduced in [HGSE23] by taking into account the distinctive structure of the QARMAv2 diffusion layer. This enhancement allows us to identify integral distinguishers that are more effective for integral key recovery attacks.

2.3 Key Recovery in Integral Attacks using the Partial-Sum Technique

For integral distinguishers derived from ZC linear hulls the sum of the outputs is zero in specific bit positions (*balanced* bit positions). We typically append some rounds to the distinguisher to build a key recovery upon an integral distinguisher. Then, we guess the involved key bits to partially decrypt the ciphertexts and compute the sum of the distinguishers' outputs in the balanced bit positions. If the sum is zero, we keep the guessed key bits as potential candidates. Otherwise, we discard the guessed key bits.

The partial-sum technique was initially introduced by Ferguson et al. in [FKL+00] to reduce the time complexity of integral attacks. Unlike the naive integral key recovery, where we guess the involved key bits all at once, the partial-sum technique divides the partial decryption into several steps. We guess a subset of the involved key bits at each step and store the intermediate results. We repeat the process until we reach the distinguisher's output. At each step, only a portion of the internal state is involved, whose values are needed to calculate the final sum. One advantage is that the size of involved positions

reduces as we approach the distinguisher's output. In addition, to compute the sum of the distinguisher's outputs in the balanced bit positions, we only need to know if each involved value appears an even or odd number of times.

Figure 6 represents the integral key recovery for 6 rounds of AES using the partial-sum technique. This attack relies on a 4-round integral distinguisher, derived by encrypting 2^{32} plaintexts that take all possible values in the main diagonal and a fixed value in other positions. After 4 rounds of AES, the sum of the outputs is zero in all bytes. The last round does not include the MixColumns, and instead of K_4, we retrieve $\bar{K}_4 = \mathrm{MC}^{-1}(K_4)$, i.e., the so-called equivalent key for 5th round. The colored numbers in Figure 6 denote the corresponding step of the partial-sum technique for each byte in the internal state or round keys. According to Figure 6 we have:

$$C_4[0] = \mathcal{S}^{-1}\Big(\bar{K}_4[0] \oplus \texttt{0E}\cdot\mathcal{S}^{-1}\left(C_6[0]\oplus K_5[0]\right) \oplus \texttt{09}\cdot\mathcal{S}^{-1}\left(C_6[7]\oplus K_5[7]\right) \\ \oplus \texttt{0D}\cdot\mathcal{S}^{-1}\left(C_6[10]\oplus K_5[10]\right) \oplus \texttt{0B}\cdot\mathcal{S}^{-1}\left(C_6[13]\oplus K_5[13]\right)\Big), \tag{5}$$

where $\mathcal{S}$ is the AES S-box.We first implement $\texttt{0E}\cdot\mathcal{S}^{-1}(.)$ by $\mathcal{S}_0(.)$, $\texttt{09}\cdot\mathcal{S}^{-1}(.)$ by $\mathcal{S}_1(.)$, $\texttt{0D}\cdot\mathcal{S}^{-1}(.)$ by $\mathcal{S}_2(.)$, and $\texttt{0B}\cdot\mathcal{S}^{-1}(.)$ by $\mathcal{S}_3(.)$ as lookup tables. Next, we perform the partial-sum key recovery as outlined in algorithm 2.

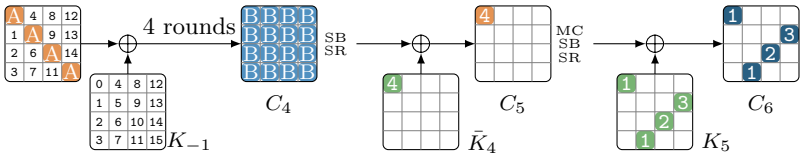


Figure 6: Involved cells in integral key recovery on 6 rounds of AES.

Algorithm 2: Partial-sum key recovery attack on 6 rounds of AES

Input: 2^{32} ciphertexts, and the set of discarded keys $\mathcal{DK}$ (empty in the first run)
Output: A set of candidates $\mathcal{K}$ for $\bar{K}_4[0]||K_5[0,7,10,13]$

$\mathcal{K} \leftarrow \emptyset$;
forall $K_5[0,7]$ **do**
  Initialize a list $\mathcal{L}_1$ of size 2^{24} with zeros;
  forall 2^{32} *ciphertexts* **do**
    $p_1 \leftarrow \mathcal{S}_0\left(C_6[0]\oplus K_5[0]\right) \oplus \mathcal{S}_1\left(C_6[7]\oplus K_5[7]\right)$;
    $\mathcal{L}_1\left[(p_1, C_6[10,13])\right] \leftarrow \mathcal{L}_1\left[(p_1, C_6[10,13])\right] \oplus 1$;
  forall $K_5[10]$ **do**
    Initialize a list $\mathcal{L}_2$ of size 2^{16} with zeros;
    forall $(p_1, C_6[10,13])$ *s.t.* $\mathcal{L}_1\left[(p_1, C_6[10,13])\right] = 1$ **do**
      $p_2 \leftarrow p_1 \oplus \mathcal{S}_2\left(C_6[10]\oplus K_5[10]\right)$;
      $\mathcal{L}_2\left[(p_2, C_6[13])\right] \leftarrow \mathcal{L}_2\left[(p_2, C_6[13])\right] \oplus 1$;
    forall $K_5[13]$ **do**
      Initialize a list $\mathcal{L}_3$ of size 2^8 with zeros;
      forall $(p_2, C_6[13])$ *s.t.* $\mathcal{L}_2\left[(p_2, C_6[13])\right] = 1$ **do**
        $p_3 \leftarrow p_2 \oplus \mathcal{S}_3\left(C_6[13]\oplus K_5[13]\right)$;
        $\mathcal{L}_3[p_3] \leftarrow \mathcal{L}_3[p_3] \oplus 1$;
      forall $\bar{K}_4[0]$ **do**
        if $\bar{K}_4[0]||K_5[0,7,10,13] \notin \mathcal{DK}$ **then**
          $\texttt{Result} \leftarrow \bigoplus_{p_3 \in \mathbf{F}_2^8:\ \mathcal{L}_3[p_3]=1} \mathcal{S}^{-1}\left(\bar{K}_4[0]\oplus p_3\right)$;
          if $\texttt{Result} = 0$ **then** $\mathcal{K} \leftarrow \mathcal{K} \cup \{\bar{K}_4[0]||K_5[0,7,10,13]\}$;
          else $\mathcal{DK} \leftarrow \mathcal{DK} \cup \{\bar{K}_4[0]||K_5[0,7,10,13]\}$
return $\mathcal{K}$;

In total 5 bytes of round keys are involved, and each balanced byte provides an 8-bit filter. Therefore, we need six sets of 2^{32} chosen plaintexts to retrieve the involved key

bits uniquely. The time complexity of the naive approach is $6 \cdot 2^{32} \cdot 2^{40} \approx 2^{74.58}$ partial decryptions. However, the time complexity of algorithm 2 is 2^{50} S-box computations. Repeating it for six sets of 2^{32} chosen plaintexts yields a total complexity of at most $6 \cdot 2^{50} \approx 2^{52.58}$ S-box lookups. The required memory to store discarded keys and also 2^{32} ciphertexts dominates the memory complexity, and the data complexity is $6 \cdot 2^{32} \approx 2^{34.58}$ chosen plaintexts. As can be seen, the partial-sum technique significantly reduces the time complexity of integral attacks. So, we use this technique to build integral key recovery attacks on `QARMAv2`.

2.4 Meet-in-the-Middle Technique

From the 6-round integral key recovery attack on `AES` one can see that the number of involved key bits to compute the final sum is an effective factor in time complexity. Meet-in-the-middle technique in integral key recovery, firstly introduced in [SW12], splits the involved key bits into two sets and enables us to retrieve each set of key bits independently. We explain this technique with a simple example. As Figure 7 shows, assume that we aim to compute $\bigoplus Z$ from the ciphertexts and check if $\bigoplus Z = 0$. In a naive approach, we must guess all the involved key bits $K_1 \cup K_2$. However, by looking at Figure 7, we observe that $Z = X \oplus Y$. Verifying $\bigoplus Z = 0$ is the same as confirming that $\bigoplus X = \bigoplus Y$. This enables us to independently calculate $\bigoplus X$ and $\bigoplus Y$ and then compare them for equality. The advantage is that we only need to guess K_1 (resp. K_2) to compute $\bigoplus X$ (resp. $\bigoplus Y$). Each guess of K_1 and K_2 that satisfies $\bigoplus X = \bigoplus Y$ is considered a potential candidate. Consequently, the time complexity of guess-and-filter for the involved key bits decreases from $2^{|K_1 \cup K_2|}$ to $2^{|K_1|} + 2^{|K_2|}$.

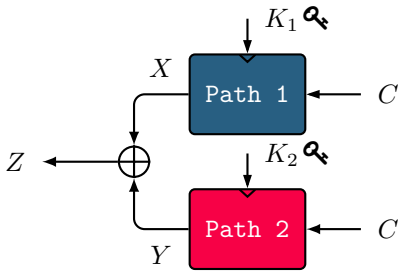


Figure 7: Meet-in-the-middle technique in integral key recovery.

3 Integral Properties of `QARMAv2` Diffusion Matrix

In this section, we show how to exploit the properties of `QARMAv2` MixColumns to bypass the diffusion layer right after the distinguisher. This way, fewer key bits will be involved in the key recovery. As a result, we can reduce the time complexity and append more rounds to the distinguisher for a key recovery attack.

The matrix used in the MixColumns of `QARMAv2` is an almost MDS matrix represented in Equation 3. We show that when two balanced cells exist in one column of a layer before the MixColumns operation, a specific linear combination of the cells at the same position after MixColumns is balanced. Let $(X_0, X_1, X_2, X_3)^T$, and $(Y_0, Y_1, Y_2, Y_3)^T$ represent

columns before and after the MixColumns, respectively. Then, we have:

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 0 & \rho & \rho^2 & \rho^3 \\ \rho^3 & 0 & \rho & \rho^2 \\ \rho^2 & \rho^3 & 0 & \rho \\ \rho & \rho^2 & \rho^3 & 0 \end{pmatrix} \times \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} \rho X_1 + \rho^2 X_2 + \rho^3 X_3 \\ \rho^3 X_0 + \rho X_2 + \rho^2 X_3 \\ \rho^2 X_0 + \rho^3 X_1 + \rho X_3 \\ \rho X_0 + \rho^2 X_1 + \rho^3 X_2 \end{pmatrix}. \tag{6}$$

Assume that $\mathbb{C}$ is a pool of ciphertexts derived from the input set of integral distinguisher. Besides, X_i, Y_j are the intermediate variables that can be expressed as a function of ciphertext and round keys. Given that the key is fixed in the integral attack, we use $X_i(c), Y_i(c)$ to indicate the dependency on the ciphertext $c \in \mathbb{C}$. Then, for $i, j \in \{0, 1, 2, 3\}$, $i \neq j$, we have

$$\bigoplus_{c\in\mathbb{C}} \left((\rho^{(i-j) \mod 4} X_i(c)) \oplus X_j(c) \right) = \bigoplus_{c\in\mathbb{C}} \left((\rho^{(i-j) \mod 4} Y_i(c)) \oplus Y_j(c) \right).$$

Hence, if X_i, and X_j have the zero-sum property, then $\bigoplus_{c\in\mathbb{C}} \rho^{(i-j) \bmod 4} Y_i(c) = \bigoplus_{c\in\mathbb{C}} Y_j(c)$, and we can use the meet-in-the-middle technique to derive $\bigoplus_{c\in\mathbb{C}} \rho^{(i-j) \bmod 4} Y_i(c)$ and $\bigoplus_{c\in\mathbb{C}} Y_j(c)$ independently. Note that we essentially bypass the diffusion effect of the MixColumns by transforming the distinguishing property after the MixColumns. With our updated model for distinguishers, as explained in Section 4, we identify new integral distinguishers for `QARMAv2` that leverage the MixColumns property to increase the number of rounds for the key recovery attack.

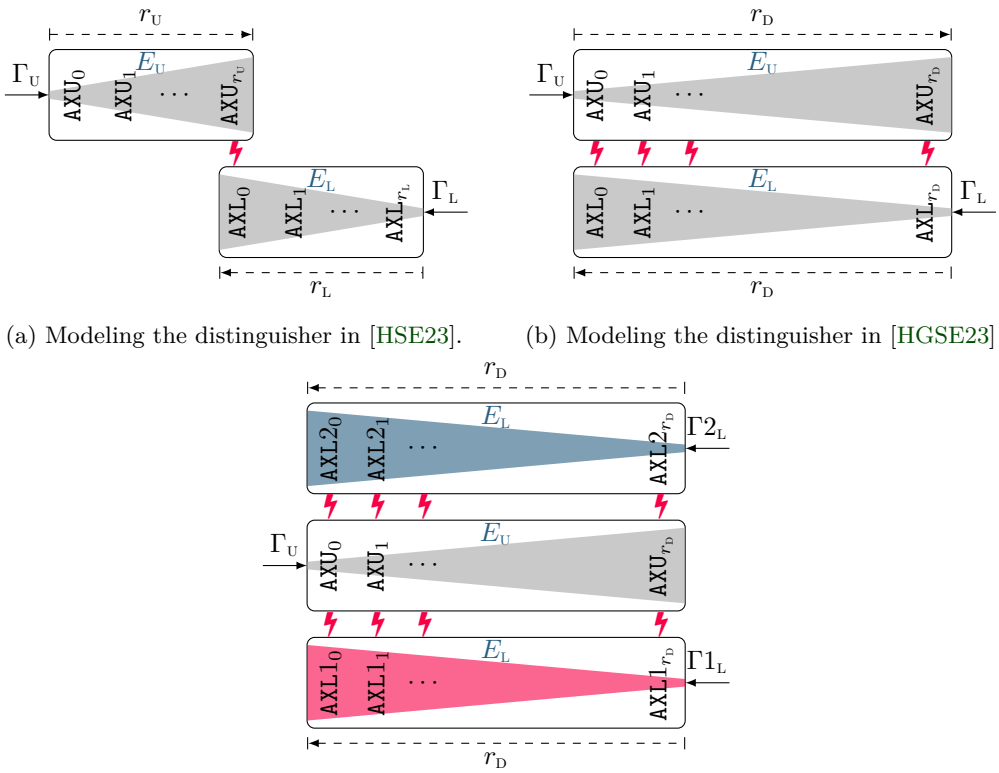
4 Search for Distinguishers

This section introduces our new CP model designed for detecting integral distinguishers in the `QARMAv2` cipher. Our approach follows the method introduced in [HSE23, HGSE23]. However, we enhance the model by considering the unique structure of the `QARMAv2` diffusion layer. This refinement enables us to identify integral distinguishers that are more effective in key recovery attacks. More precisely, we take advantage of the properties of the MixColumns matrix discussed in Section 3 to bypass the diffusion effect of the MixColumns layer and reduce the number of involved key bits in the key recovery attack. We elaborate on our model for a TBC following the `TWEAKEY` framework, as represented in Figure 5. Therefore, besides `QARMAv2`, it is adaptable to many TBCs, such as `SKINNY`, `MANTIS`, and `CRAFT`, which share similar diffusion layers.

We aim to use Theorem 2 to find a ZC distinguisher suitable for integral key recovery attacks. Then, using Theorem 1, we convert the ZC distinguisher to an integral distinguisher and build a key recovery attack. Therefore, we explain how to create a CP model to search for ZC distinguishers based on Theorem 2. For this purpose, we must encode deterministic linear trails forward and backward through the cipher. We model the deterministic linear trails using the same method as in [HSE23, HGSE23]. Thus, one can refer to [HSE23, HGSE23] for more details on encoding deterministic linear trails at the cell level.

We first briefly review the model in [HGSE23] for finding integral distinguishers. As illustrated in Figure 5, let E be a tweakable block cipher following the STK framework, with block size of $n = m \cdot c$, where m and c denote the number of cells and the cell size, respectively. Suppose that E has z parallel independent tweakey paths, and h denotes the permutation on the position of the tweakey cells. Besides, assume that $STK_r[i]$ represents the ith cell of the subtweakey after r rounds.

As Figure 8b illustrates, we define integer variables $\texttt{AXU}_r[i]$ (resp. $\texttt{AXL}_r[i]$) to represent the activeness pattern of the ith cell of the internal state after r rounds in the forward direction (resp. background direction). The domain of these variables is $\{0, 1, 2, 3\}$, where



(a) Modeling the distinguisher in [HSE23].

(b) Modeling the distinguisher in [HGSE23]

(c) Our modeling for integral distinguishers.

Figure 8: Modeling the ZC and integral distinguishers as a CSP problem.

0 indicates the zero linear mask, 1 indicates a fixed nonzero linear mask, 2 indicates a free nonzero linear mask, and 3 indicates a free linear mask. Then, as visualized by Figure 8b, we define some constraints to encode the propagation of the deterministic linear trails in forward and backward direction over r_D rounds independently. For more details on the constraints, please refer to [HSE23, HGSE23]. We also add the constraints $\sum_{i=0}^{m-1} \mathtt{AXU}_0[i] \neq 0$, and $\sum_{i=0}^{m-1} \mathtt{AXL}_{r_D}[i] \neq 0$ to exclude trivial solutions. Assume that $\mathit{CSP}_U(\mathtt{AXU}_0, \ldots, \mathtt{AXU}_{r_D})$ and $\mathit{CSP}_L(\mathtt{AXL}_0, \ldots, \mathtt{AXL}_{r_D})$ denote the CSP models for the forward and backward propagation, respectively.

We define the integer variables $\mathtt{ASTK}_r[i] \in \{0, 1, 2, 3\}$ to encode the activation pattern of $STK_r[i]$. We know that the activeness pattern of tweakey cells in the propagation of linear trails should follow the linear mask of the internal states. Assume that $\mathtt{AYU}$ and $\mathtt{AYL}$ are integer variables like $\mathtt{AXU}$ and $\mathtt{AXL}$, indicating the activeness pattern of the internal state right before the round tweakey addition. Therefore, we add the new constraint $\mathtt{ASTK}_r[i] = \min\{\mathtt{AYU}_r[i], \mathtt{AYL}_r[i]\}$ for all $0 \leq r \leq r_D - 1$ and $0 \leq i \leq m - 1$, to link the activeness pattern of the subtweakey to the activeness pattern of the internal state. This way, the subtweakey follows the activeness pattern in one of the forward or backward propagations with less active cells. Then, to ensure that the conditions of Theorem 2 are met, we add the following constraint:

$$\mathit{CSP}_{TK}(\mathtt{ASTK}_0, \ldots, \mathtt{ASTK}_{r_D-1}) :=$$
$$\bigvee_{i=0}^{m-1} \left(\left(\begin{array}{l} \sum_{r=0}^{r_D-1} \mathtt{bool2int}\left(\mathtt{ASTK}_r[h^{-r}(i)] \neq 0\right) \leq z \\ \wedge \bigvee_{r=0}^{r_D-1} \left(\mathtt{ASTK}_r[h^{-r}(i)] = 1\right) \end{array} \right) \vee \left(\bigwedge_{r=0}^{r_D-1} \mathtt{ASTK}_r[h^{-r}(i)] = 0 \right) \right) \tag{7}$$

The conjunction of the CSP models above, i.e., $\mathit{CSP}_{\text{D}} = \mathit{CSP}_{\text{U}} \wedge \mathit{CSP}_{TK} \wedge \mathit{CSP}_{\text{L}}$, creates a unified CP/MILP model based on satisfiability, whose all feasible solutions are the ZC/integral distinguishers for r_{D} rounds of the block cipher E. By including the objective function $\max \sum_{i=0}^{m-1} \texttt{AXU}_0[i]$, we can maximize the number of linearly active cells at the input. According to Theorem 1, the number of active cells at the input of the corresponding integral distinguisher is minimized, and we can find integral distinguishers with minimum data complexity. Additionally, the linear combination $\bigoplus_{i\in\{0,\cdots,m-1\}:\texttt{AXL}_{r_{\text{D}}}[i]\neq 0} X_{r_{\text{D}}}[i]$ is balanced (has a zero-sum property).

The authors of [HGSE23] applied the model above to find integral distinguishers for some TBCs, including `QARMAv2`, resulting in a significant improvement for integral distinguishers of `QARMAv2`. However, all integral distinguishers reported in [HGSE23] have only one balanced cell at the output right before the MixColumns. Therefore, we can not exploit the MixColumns property of `QARMAv2` because we need at least two balanced cells in one column to bypass the diffusion effect of the MixColumns layer. In what follows, we explain how to refine the model in [HSE23, HGSE23] to find integral distinguishers with more than one balanced cell in one column at the output of distinguishers.

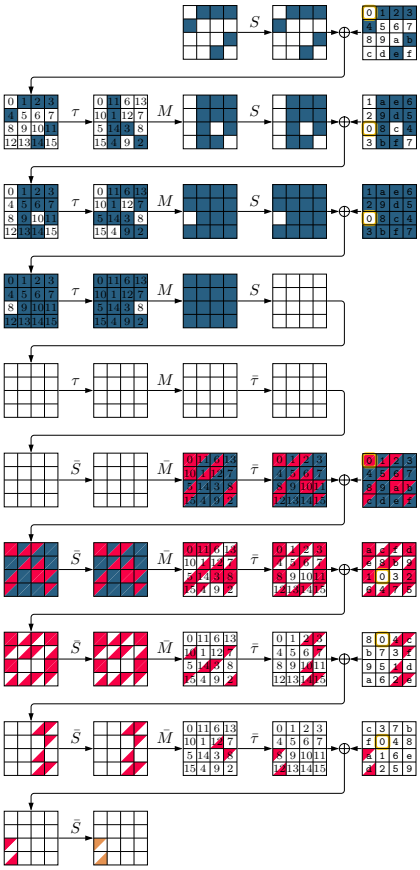
We need at least two balanced cells in the output of distinguishers to exploit the MixColumns property. We keep the constraints for the forward propagation, i.e., CSP_{U}, unchanged. However, we modify the model for the backward propagation. As illustrated in Figure 8c, we create two independent CSP models $\mathit{CSP}1_{\text{L}}$, and $\mathit{CSP}2_{\text{L}}$ with independent variables, $(\texttt{AXL}1_r[i], \texttt{AYL}1_r[i])$, and $(\texttt{AXL}2_r[i], \texttt{AYL}2_r[i])$, respectively, to model the deterministic linear trails in the backward direction. The idea is that the combination of each CSP model $\mathit{CSP}1_{\text{L}}$ and $\mathit{CSP}2_{\text{L}}$ with CSP_{U} should create a CSP model whose solutions are ZC/integral distinguishers. We define two independent sets of integer variables $\texttt{ASTK}1_r[i]$, and $\texttt{ASTK}2_r[i]$ for subtweakey cells and add the following constraints to link the activeness pattern of the subtweakey to the activeness pattern of the internal state for each backward propagation: $\texttt{ASTK}1_r[i] = \min\{\texttt{AYU}_r[i], \texttt{AYL}1_r[i]\}$, $\texttt{ASTK}2_r[i] = \min\{\texttt{AYU}_r[i], \texttt{AYL}2_r[i]\}$. Then, to ensure that the conditions of Theorem 2 hold for both CSP models $\mathit{CSP}_{\text{U}} \wedge \mathit{CSP}1_{\text{L}}$ and $\mathit{CSP}_{\text{U}} \wedge \mathit{CSP}2_{\text{L}}$, we include the constraints CSP_{DTK} as follows:

$$\texttt{contradict1}[i] := \begin{pmatrix} \sum_{r=0}^{r_{\text{D}}-1} \mathit{bool2int}\left(\texttt{ASTK}1_r[h^{-r}(i)] \neq 0\right) \leq z \\ \wedge \ \bigvee_{r=0}^{r_{\text{D}}-1} \left(\texttt{ASTK}1_r[h^{-r}(i)] = 1\right) \end{pmatrix} \vee \left(\bigwedge_{r=0}^{r_{\text{D}}-1} \texttt{ASTK}1_r[h^{-r}(i)] = 0\right) \quad (8)$$

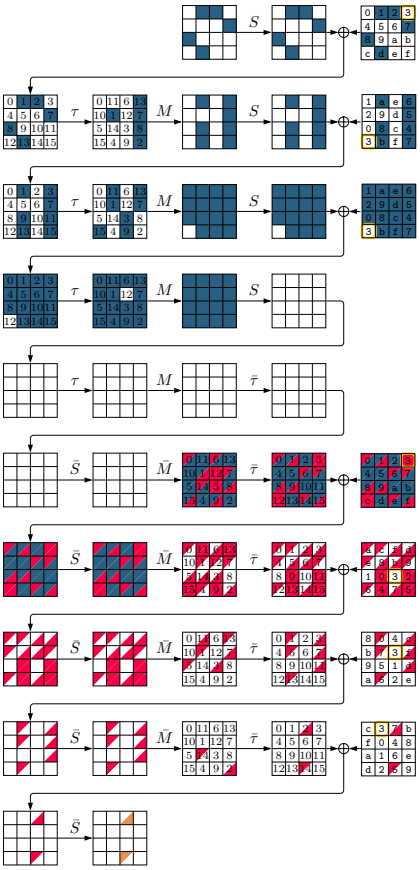
$$\texttt{contradict2}[i] := \begin{pmatrix} \sum_{r=0}^{r_{\text{D}}-1} \mathit{bool2int}\left(\texttt{ASTK}2_r[h^{-r}(i)] \neq 0\right) \leq z \\ \wedge \ \bigvee_{r=0}^{r_{\text{D}}-1} \left(\texttt{ASTK}2_r[h^{-r}(i)] = 1\right) \end{pmatrix} \vee \left(\bigwedge_{r=0}^{r_{\text{D}}-1} \texttt{ASTK}2_r[h^{-r}(i)] = 0\right) \quad (9)$$

$$\bigvee_{i=0}^{m-1} (\texttt{contradict1}[i] + \texttt{contradict2[i]} = 2) = \mathit{True}, \quad (10)$$

where $\texttt{contradict1}[i]$, and $\texttt{contradict2}[i]$ are binary variables for all $0 \leq i \leq m-1$. We want to ensure that the $\mathit{CSP}1_{\text{L}}$ and $\mathit{CSP}2_{\text{L}}$ yield two different activeness patterns for the output cells. Additionally, the active cells in $\texttt{AXL}1_{r_{\text{D}}}$ and $\texttt{AXL}2_{r_{\text{D}}}$ should be in the same columns. For this purpose, we first limit the value of $\texttt{AXL}1_{r_{\text{D}}}[i], \texttt{AXL}2_{r_{\text{D}}}[i]$ to $\{0, 1\}$. Then we include the constraints $\texttt{AXL}1_{r_{\text{D}}}[i] \neq \texttt{AXL}2_{r_{\text{D}}}[i]$ for all $0 \leq i \leq m-1$. Finally, we add some constraints to ensure that the active cells of $\texttt{AXL}1_{r_{\text{D}}}$ and $\texttt{AXL}2_{r_{\text{D}}}$ appear in the same column. Therefore, the conjunction of the CSP models $\mathit{CSP}_{\text{D}} := \mathit{CSP}_{\text{U}} \wedge \mathit{CSP}1_{\text{L}} \wedge \mathit{CSP}2_{\text{L}} \wedge \mathit{CSP}_{DTK}$ creates a unified CP/MILP model based on satisfiability, whose all feasible solutions are the integral distinguishers for r_{D} rounds of the block cipher E with at least two balanced cells in the same column at the output of distinguishers. We developed this model for all versions of `QARMAv2` using MiniZinc [NSB+07] and successfully solved it with the open-source CP solver Or-Tools [PF] on a regular laptop within a matter of seconds.



(a) Integral distinguisher I for 9 rounds of QARMAv2-64. Data complexity: 2^{44}.



(b) Integral distinguisher II for 9 rounds of QARMAv2-64. Data complexity: 2^{48}.

Figure 9: ZC-based integral distinguishers for QARMAv2-64 ($\mathscr{T} = 1$).

For QARMAv2-64 ($\mathscr{T} = 1/2$), and QARMAv2-128 ($\mathscr{T} = 2$), we found 9/10-round, and 11-round ZC-based integral distinguishers with data complexity 2^{44}, respectively. Figure 9, Figure 10, Figure 11 illustrate some of these distinguishers featuring two balanced cells within a single column of the output state. The colors employed in the figures signify the activity pattern of the corresponding cells in the ZC distinguishers, where ■ denotes any linear mask, ■ signifies a nonzero linear mask, and ■ represents a nonzero fixed linear mask. Inactive cells remain white (blank). To transform the ZC distinguishers into integral distinguishers, we must invert the activity pattern in the input. In other words, active cells with an arbitrary linear mask (■) should assume a fixed value, while inactive cells should take all possible value exactly once. Then, the output cells with ■ are balanced in the corresponding integral distinguisher.

As mentioned earlier, we model the propagation of linear masks in the backward direction using two independent CSP models. This is why we depict the activity pattern in the backward direction with upper and lower triangles in each cell. For instance, ◪ and ◪ signify that the linear mask of the corresponding cell can assume any value in one backward path, but it must be nonzero in the other backward path. In addition, ▢ denotes the tweak cell that should take all possible values in the corresponding integral distinguishers.

We explain the interpretation of Figure 9 as an example, and interpreting other figures is similar. Figure 9 represents a 9-round ZC linear hull for QARMAv2, taking the tweak

(a) Integral distinguisher I for 10 rounds of QARMAv2-64. Data complexity: 2^{44}.

(b) Integral distinguisher II for 10 rounds of QARMAv2-64. Data complexity: 2^{44}.

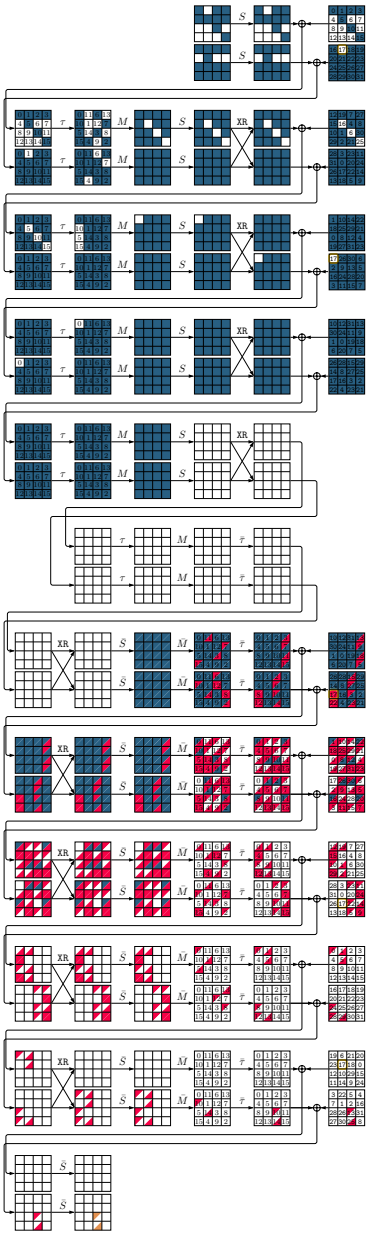
Figure 10: ZC-based integral distinguishers for QARMAv2-64 ($\mathscr{T} = 2$).

schedule into account. As seen in Figure 9a, 6 input cells can take any linear mask (■), while the linear masks of the other 10 input cells are zero. The output cells can take a fixed nonzero linear mask (■) in 8th (first backward propagation ◪) and 12th (second backward propagation ◪) cells. The tweak cell 0 takes a nonzero liner mask exactly once (after the reflector construction), whereas its linear mask is zero everywhere else. As a result, according to Theorem 2, we have two independent ZC linear hulls with the same activeness pattern at the input, but different active cells at the same column of the output.

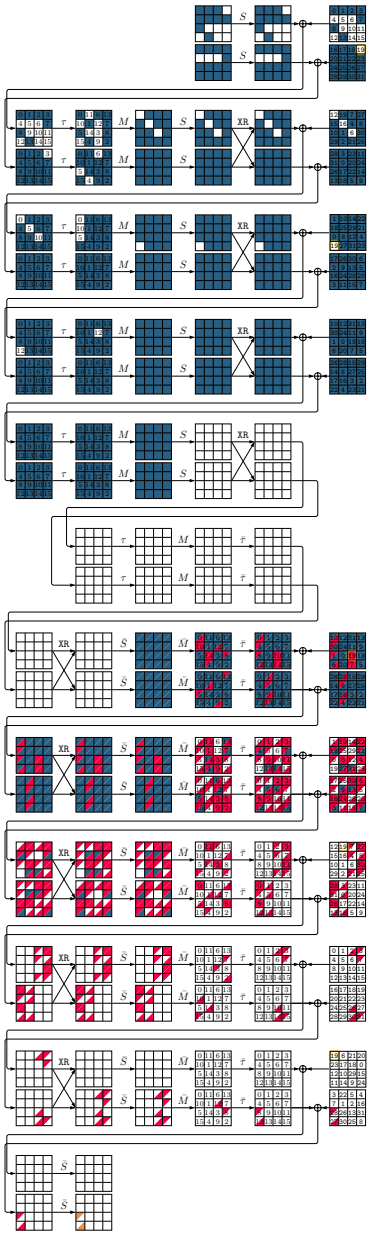
To convert the ZC linear hulls in Figure 9 into integral distinguishers, the input cells with active linear masks (■) should take a fixed value and the linearly inactive cells should take all possible values exactly once. In addition, the tweak cell number 0 should take all possible values exactly once. Then, the output cells with ◪/◪ labels are balanced in the corresponding integral distinguishers. Due to $10 + 1$ active cell (10 active cells at the internal state and 1 active cell in the tweak) at the input, the data complexity of the resulting integral distinguisher is 2^{44}.

According to Table 2, the data complexity of any valid attack on QARMAv2-64 (resp. QARMAv2-128) should be less than 2^{56} (resp. 2^{80}). Therefore, our integral distinguishers satisfy the data complexity limits. We also found a 12-round integral distinguisher for

QARMAv2-128 ($\mathscr{T} = 2$) with two balanced output cells, as illustrated in Figure 15. However, we do not use it in our key recovery attack since its data complexity is 2^{96} (above the threshold). In Section 5, we elaborate on applying meet-in-the-middle and partial-sum techniques to construct an efficient key recovery attack based on our new integral distinguishers.



(a) Integral distinguisher I for 11 rounds of QARMAv2-128. Data complexity: 2^{44}.



(b) Integral distinguisher II for 11 rounds of QARMAv2-128. Data complexity: 2^{44}.

Figure 11: ZC-based integral distinguishers for QARMAv2-128 ($\mathscr{T} = 2$).

5 Integral Key Recovery

Here, we use the partial-sum technique [FKL+00] and the meet-in-the-middle approach [SW12] to provide key recovery attacks upon our distinguishers for QARMAv2. Moreover, to exploit the low data complexity of the integral distinguisher, we construct each distillation table after guessing the whole of L_0. It is also helpful to reduce the required memory complexity for the meet-in-the-middle approach.

Recall that the authors of QARMAv2 claimed $\kappa - \varepsilon$-bit security for κ-bit secret key, where ε is adjusted to values such as 16 for standardization purposes. We emphasize that our key recovery attacks are valid for the parameters suggested for general or standardization purposes.

5.1 Integral Attack for 13-Round QARMAv2-64 ($\mathscr{T} = 1$)

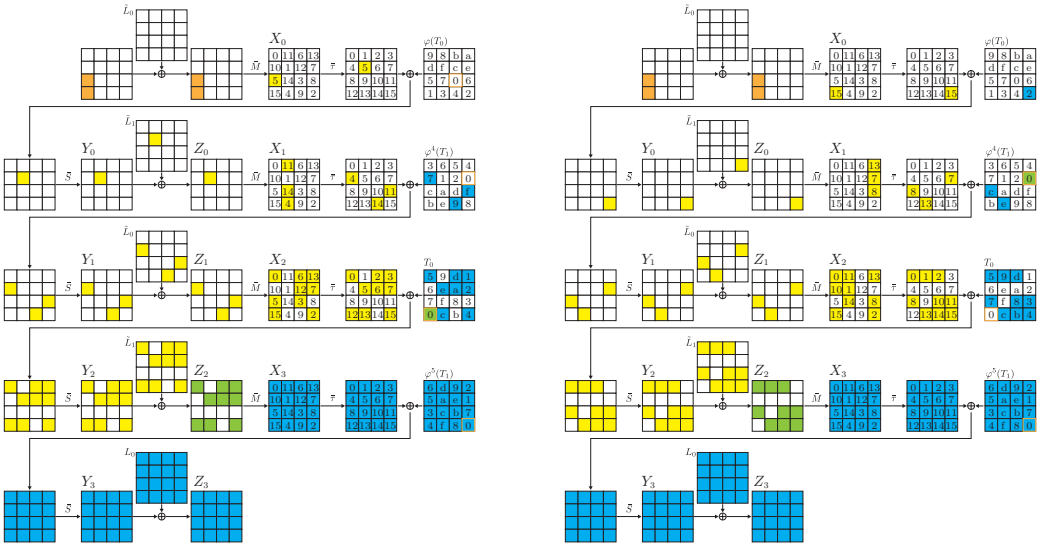


Figure 12: Key recovery for 13-round QARMAv2-64 ($\mathscr{T} = 1$)

Here, we propose an integral attack against 13-round of QARMAv2-64 ($\mathscr{T} = 1$) by appending 4 rounds for key recovery to the ciphertext side of our 9-round distinguisher in Figure 9a. Figure 12 shows the overview of the key recovery. As seen in Figure 12, thanks to having two balanced cells in the same column at the output of the distinguisher, we can bypass the diffusion effect of the MixColumns layer. Otherwise, much more key bits would be involved in the key recovery, yielding a higher time complexity. Besides, we sometimes guess $\tilde{L}_0 = M \circ \tau(L_0)$ and $\tilde{L}_1 = M \circ \tau(L_1)$ instead of L_0 and L_1. Let X_i, Y_i, and Z_i be internal states defined in Figure 12.

Attack Procedure. The 9-round integral distinguisher is built by using 2^{44} chosen plaintexts. We have the 8-bit balanced value after the S-box layer. We use the following relation for the efficiency of the key recovery.

$$\begin{cases} \bigoplus Z_{-1}[8] = \bigoplus \left(\rho^2(X_0[0]) \oplus \rho^3(X_0[10]) \oplus \rho(X_0[15])\right) = 0, \\ \bigoplus Z_{-1}[12] = \bigoplus \left(\rho(X_0[0]) \oplus \rho^2(X_0[10]) \oplus \rho^3(X_0[5])\right) = 0, \end{cases}$$
$$\Rightarrow \bigoplus \left(\rho(X_0[15]) \oplus X_0[5]\right) = \bigoplus \left(Z_{-1}[8] \oplus \rho(Z_{-1}[12])\right) = 0.$$

Table 3: Partial-sum technique to compute $X_0[5]$, where L_0 is guessed in advance.

Step	Guessed key	Stored nibbles	(size)	Complexity (unit)
0	-	$Z_2[0,2,3,5,6,7,11,12,13], T_0[0]$	2^{40}	
1	$\tilde{L}_1[12]$	$Z_2[0,2,3,5,6,7,11,13], X_2[12]$	2^{36}	$2^4 \times 2^{40} \times 1$ (SB)
2	$\tilde{L}_1[3]$	$Z_2[0,2,5,6,7,13,15], \mathtt{mix}(X_2[12], X_2[3])$	2^{32}	$2^4 \times 2^{36} \times 2^4$ (SB)
3	$\tilde{L}_1[6]$	$Z_2[0,2,5,7,13,15], Z_1[14]$	2^{28}	$2^4 \times 2^{32} \times 2^8$ (SB)
4	$(\tilde{L}_0[14])$	$Z_2[0,2,5,7,13,15], X_1[14]$	2^{28}	$1 \times 2^{28} \times 2^{12}$ (SB)
5	$\tilde{L}_1[0]$	$Z_2[2,5,7,13,15], X_2[0], X_1[14]$	2^{28}	$2^4 \times 2^{28} \times 2^{12}$ (SB)
6	$\tilde{L}_1[5]$	$Z_2[2,7,13,15], \mathtt{mix}(X_2[0], X_2[5]), X_1[14]$	2^{24}	$2^4 \times 2^{28} \times 2^{16}$ (SB)
7	$\tilde{L}_1[15]$	$Z_2[2,7,13], Z_1[4], X_1[14]$	2^{20}	$2^4 \times 2^{24} \times 2^{20}$ (SB)
8	$(\tilde{L}_0[4])$	$Z_2[2,7,13], \mathtt{mix}(X_1[4], X_1[14])$	2^{16}	$1 \times 2^{20} \times 2^{24}$ (SB)
9	$\tilde{L}_1[2]$	$Z_2[7,13], X_2[2], \mathtt{mix}(X_1[4], X_1[14])$	2^{16}	$2^4 \times 2^{16} \times 2^{24}$ (SB)
10	$\tilde{L}_1[7]$	$Z_2[13], \mathtt{mix}(X_2[2], X_2[7]), \mathtt{mix}(X_1[4], X_1[14])$	2^{12}	$2^4 \times 2^{16} \times 2^{28}$ (SB)
11	$\tilde{L}_1[13]$	$Z_1[11], \mathtt{mix}(X_1[4], X_1[14])$	2^8	$2^4 \times 2^{12} \times 2^{32}$ (SB)
12	$(\tilde{L}_0[11])$	$Z_0[5]$	2^4	$1 \times 2^8 \times 2^{36}$ (SB)
13	$(\tilde{L}_1[5])$	$X_0[5]$	2^4	$1 \times 2^4 \times 2^{36}$ (SB)
Total				$2^{50.15}$ (SB)

Only two nibbles of X_0 are enough to observe the 4-bit balanced property. Therefore, we use the meet-in-the-middle approach, where we independently compute the sum of $X_0[5]$ and $X_0[15]$. We finally retrieve the secret key satisfying $\bigoplus X_0[5] = \rho \bigoplus X_0[15]$. One structure, using 2^{44} chosen plaintexts, can be a 4-bit filter, i.e., the secret-key space is reduced by the factor of 2^{-4}. However, we need a more substantial filtering effect to build an attack whose complexity is less than 2^{128-16}, and hence valid for standardization purposes. Therefore, we use s structures to enhance it to a $4s$-bit filter.

The straightforward meet-in-the-middle approach yields an enormous memory complexity to store all the guessed key bits. To reduce the memory complexity, we share some guesses, specifically the whole of L_0, in both procedures. Specifically, we use the following procedure.

1. Guess the whole of L_0, 64 bits, and construct two distillation tables to compute the sum of $X_0[5]$ and $X_0[15]$.
   (a) Compute the sum of $X_0[5]$ by using the partial-sum technique (see Table 3).
   (b) Compute the sum of $X_0[15]$ by using the partial-sum technique (see Table 4).
   (c) Apply the meet-in-the-middle approach and retrieve about 2^{64-4s} key candidates about L_1.
   (d) Guess 2^{64-4s} L_1 and check the correctness by a few trial encryptions.

Table 3 and Table 4 summarize the partial-sum procedures to compute the sum of $X_0[5]$ and the sum of $X_0[15]$, respectively. Here, $\mathtt{mix}(X, X')$ denotes a linear function represented by $\rho^i(X) \oplus \rho^j(X')$ with a proper i and j.

We finally estimate the attack complexity. The time complexity is

$$2^{64} \times \left(s \times 2^{44}\mathtt{RF} + s \times 2^{50.15}\mathtt{SB} + s \times 2^{50.67}\mathtt{SB} + 2^{64-4s}\mathtt{ENC}\right),$$

Note that each list for the meet-in-the-middle contains 2^{36} key candidates. Even if we repeat the procedure 2^{64} times for guessing L_0, the cost of sorting and matching two lists is negligible. When we regard SB and RF as $\frac{1}{16}$RF and $\frac{1}{13}$ENC, respectively, the total time complexity is $2^{110.47}$ with $s = 5$. Thus, the data complexity is 5×2^{44}. Each partial-sum procedure has to store at most 2^{44} s-bit values, but storing $s \times 2^{44}$ ciphertexts is more significant. Therefore, the memory complexity is about $s \times 2^{44}$.

Table 4: Partial-sum technique for $X_0[15]$, where L_0 is guessed in advance.

Step	Guessed key	Stored data nibbles	(size)	Complexity (unit)
0	-	$Z_2[0,1,8,10,11,13,14,15], T_0[0]$	2^{40}	
1	$\tilde{L}_1[2]$	$Z_2[0,1,8,10,11,13,14,15], T_0[0], X_2[2]$	2^{40}	$2^4 \times 2^{40} \times 1$ (SB)
2	$\tilde{L}_1[8]$	$Z_2[0,1,10,11,13,14,15], T_0[0], \texttt{mix}(X_2[2], X_2[8])$	2^{36}	$2^4 \times 2^{40} \times 2^4$ (SB)
3	$\tilde{L}_1[13]$	$Z_2[0,1,10,11,14,15], T_0[0], Z_1[7]$	2^{32}	$2^4 \times 2^{36} \times 2^8$ (SB)
4	$(\tilde{L}_0[11])$	$Z_2[0,1,10,11,14,15], X_1[7]$	2^{28}	$1 \times 2^{32} \times 2^{12}$ (SB)
5	$\tilde{L}_1[0]$	$Z_2[1,10,11,14,15], X_2[0], X_1[7]$	2^{28}	$2^4 \times 2^{28} \times 2^{12}$ (SB)
6	$\tilde{L}_1[10]$	$Z_2[1,11,14,15], \texttt{mix}(X_2[0], X_2[10]), X_1[7]$	2^{24}	$2^4 \times 2^{28} \times 2^{16}$ (SB)
7	$\tilde{L}_1[15]$	$Z_2[1,11,14], Z_1[8], X_1[7]$	2^{20}	$2^4 \times 2^{24} \times 2^{20}$ (SB)
8	$(\tilde{L}_0[8])$	$Z_2[1,11,14], \texttt{mix}(X_1[7], X_1[8])$	2^{16}	$1 \times 2^{20} \times 2^{24}$ (SB)
9	$\tilde{L}_1[1]$	$Z_2[11,14], X_2[1], \texttt{mix}(X_1[7], X_1[8])$	2^{16}	$2^4 \times 2^{16} \times 2^{24}$ (SB)
10	$\tilde{L}_1[11]$	$Z_2[14], \texttt{mix}(X_2[1], X_2[11]), \texttt{mix}(X_1[7], X_1[8])$	2^{12}	$2^4 \times 2^{16} \times 2^{28}$ (SB)
11	$\tilde{L}_1[14]$	$Z_1[13], \texttt{mix}(X_1[7], X_1[8])$	2^8	$2^4 \times 2^{12} \times 2^{32}$ (SB)
12	$(\tilde{L}_0[13])$	$Z_0[15]$	2^4	$1 \times 2^8 \times 2^{36}$ (SB)
13	$(\tilde{L}_1[15])$	$X_0[15]$	2^4	$1 \times 2^4 \times 2^{36}$ (SB)
Total				$2^{50.67}$ (SB)

5.2 Integral Attack for 14-Round QARMAv2-64 ($\mathscr{T} = 2$)

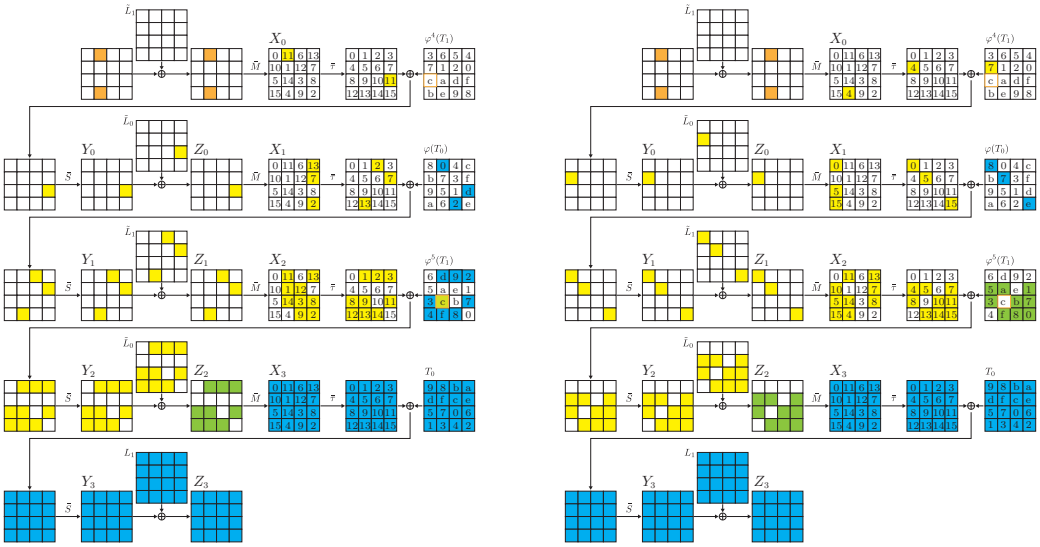


Figure 13: Key recovery for 14-round QARMAv2-64 ($\mathscr{T} = 2$)

We have discovered a 10-round integral distinguisher with two balanced output words for QARMAv2-64 ($\mathscr{T} = 2$) as illustrated in Figure 10. By adding 4 rounds to the integral distinguisher, we obtain a 14-round integral attack. Almost the same attack procedure as the key recovery for $\mathscr{T} = 1$ is applicable. We guess the whole of L_1 and construct two distillation tables for computing the sums of $X_0[11]$ and $X_0[4]$. The complexity for computing the sum of $X_0[4]$ is slightly more efficient because it involves no active tweak nibble. Finally, the complexity is

$$2^{64} \times \left(s \times 2^{44}\texttt{RF} + s \times 2^{50.15}\texttt{SB} + s \times 2^{50.13}\texttt{SB} + 2^{64-4s}\texttt{ENC}\right),$$

where RF denotes the cost of the round function, SB denotes the cost of the S-box, and ENC represents the cost of the encryption algorithm. With the same conversion of the unit of complexity, the attack complexity is $2^{110.17}$ with $s = 5$. The required data complexity is 5×2^{44}. Figure 13 summarizes the 14-round key recovery.

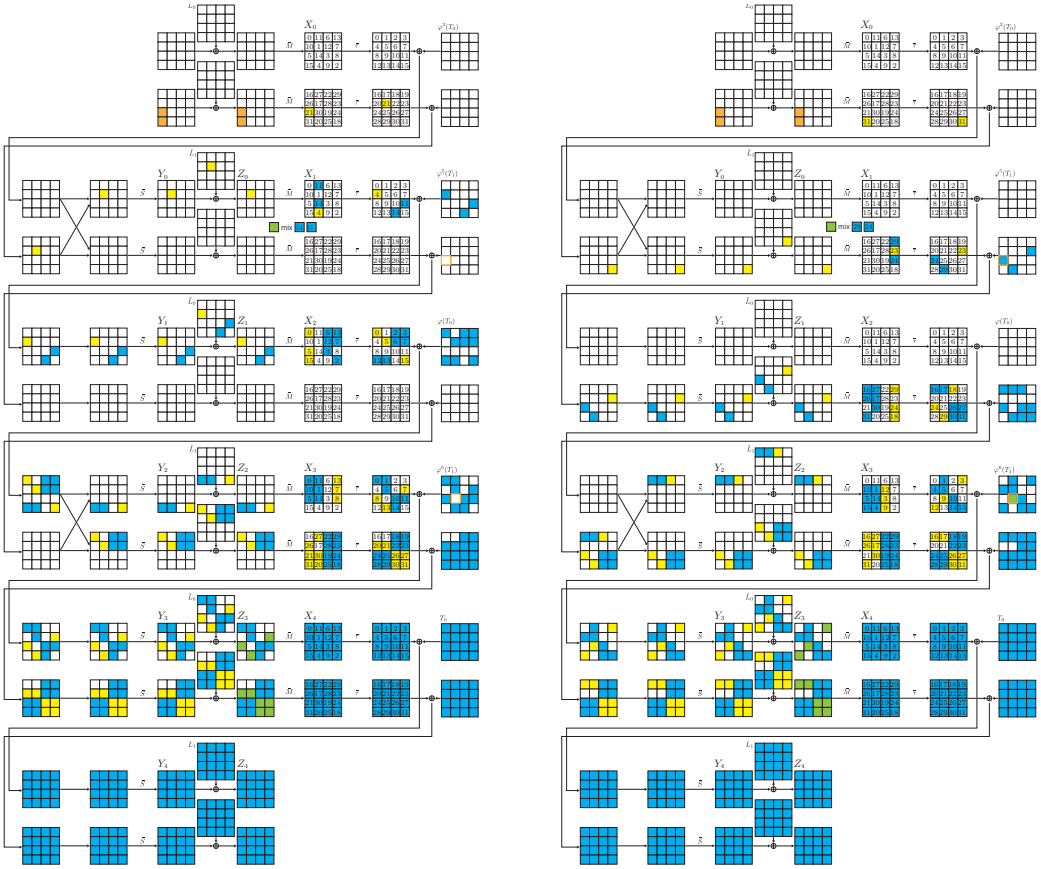


Figure 14: Key recovery for 16-round QARMAv2-128 ($\mathscr{T} = 2$)

5.3 Integral Attack for 16-Round QARMAv2-128 ($\mathscr{T} = 2$)

We append 5 rounds to our 11-round integral distinguisher for QARMAv2-128 ($\mathscr{T} = 2$) in Figure 11 to obtain a 16-round integral attack. The attack procedure is similar to the QARMAv2-64 attack. We initially guess further bits to reduce the time and memory complexity, i.e., the whole of L_1 and part of $\tilde{L}_0$. Specifically, we use the following procedure.

1. Guess the whole of L_1 and $\tilde{L}_0[1, 5, 10, 14, 18, 19, 22, 23, 24, 25, 28, 29]$, 176 bits and compute 2^{44} partial decryption.
   (a) Guess $\tilde{L}_0[0, 11]$ and construct a distillation table to compute the sum of $X_0[21]$. Compute the sum of $X_0[21]$ by using the partial-sum technique (see Table 5).
   (b) Guess $\tilde{L}_0[4, 15]$ and construct a distillation table to compute the sum of $X_0[31]$. Compute the sum of $X_0[31]$ by using the partial-sum technique (see Table 6).
   (c) Apply the meet-in-the-middle approach and retrieve about 2^{80-4s} key candidates about L_0.
   (d) Guess 2^{80-4s} L_0 and check the correctness by a few trial encryptions.

Table 5 and Table 6 summarizes the partial-sum procedures to compute the sum of $X_0[21]$ and the sum of $X_0[31]$, respectively. Here, $\texttt{mix}(X, X')$ denotes a linear function represented by $\rho^i(X) \oplus \rho^j(X')$ with a proper i and j.

The time complexity is

$$2^{176} \times \left(s \times 2^{44}\texttt{PD} + s \times 2^8 \times 2^{54.15}\texttt{SB} + s \times 2^8 \times 2^{54.15}\texttt{SB} + 2^{80-4s}\texttt{ENC}\right),$$

Table 5: Partial-sum technique to compute $X_0[21]$, where L_1 and $\tilde{L}_0[0, 1, 5, 10, 11, 14, 18, 19, 22, 23, 24, 25, 28, 29]$ are guessed in advance.

Step	Guessed key	Stored nibbles	(size)	Complexity (unit)
1	$\tilde{L}_0[7]$	$Z_3[8, 13, 20, 21, 26, 27, 30, 31], X_3[7], \texttt{mix}(X_1[11], X_1[14])$	2^{40}	$2^4 \times 2^{40} \times 1$ (SB)
2	$\tilde{L}_0[8]$	$Z_3[13, 20, 21, 26, 27, 30, 31], \texttt{mix}(X_3[7], X_3[8]), \texttt{mix}(X_1[11], X_1[14])$	2^{36}	$2^4 \times 2^{40} \times 2^4$ (SB)
3	$\tilde{L}_0[13]$	$Z_3[20, 21, 26, 27, 30, 31], Z_2[15], \texttt{mix}(X_1[11], X_1[14])$	2^{32}	$2^4 \times 2^{36} \times 2^8$ (SB)
4	$(\tilde{L}_1[15])$	$Z_3[20, 21, 26, 27, 30, 31], X_2[15], \texttt{mix}(X_1[11], X_1[14])$	2^{32}	$1 \times 2^{32} \times 2^{12}$ (SB)
5	$\tilde{L}_0[21]$	$Z_3[20, 26, 27, 30, 31], X_3[21], X_2[15], \texttt{mix}(X_1[11], X_1[14])$	2^{32}	$2^4 \times 2^{32} \times 2^{12}$ (SB)
6	$\tilde{L}_0[26]$	$Z_3[20, 27, 30, 31], \texttt{mix}(X_3[21], X_3[26]), X_2[15], \texttt{mix}(X_1[11], X_1[14])$	2^{28}	$2^4 \times 2^{32} \times 2^{16}$ (SB)
7	$\tilde{L}_0[31]$	$Z_3[20, 27, 30], Z_2[16], X_2[15], \texttt{mix}(X_1[11], X_1[14])$	2^{24}	$2^4 \times 2^{28} \times 2^{20}$ (SB)
8	$(\tilde{L}_1[16])$	$Z_3[20, 27, 30], \texttt{mix}(X_2[0], X_2[15]), \texttt{mix}(X_1[11], X_1[14])$	2^{20}	$1 \times 2^{24} \times 2^{24}$ (SB)
9	$\tilde{L}_0[20]$	$Z_3[27, 30], X_3[20], \texttt{mix}(X_2[0], X_2[15]), \texttt{mix}(X_1[11], X_1[14])$	2^{20}	$2^4 \times 2^{20} \times 2^{24}$ (SB)
10	$\tilde{L}_0[27]$	$Z_3[30], \texttt{mix}(X_3[20], X_3[27]), \texttt{mix}(X_2[0], X_2[15]), \texttt{mix}(X_1[11], X_1[14])$	2^{16}	$2^4 \times 2^{20} \times 2^{28}$ (SB)
11	$\tilde{L}_0[30]$	$Z_2[21], \texttt{mix}(X_2[0], X_2[15]), \texttt{mix}(X_1[11], X_1[14])$	2^{12}	$2^4 \times 2^{16} \times 2^{32}$ (SB)
12	$(\tilde{L}_1[21])$	$Z_1[4], \texttt{mix}(X_1[11], X_1[14])$	2^8	$1 \times 2^{12} \times 2^{36}$ (SB)
13	$\tilde{L}_0[4]$	$Z_0[5]$	2^4	$2^4 \times 2^8 \times 2^{36}$ (SB)
14	$(\tilde{L}_1[5])$	$X_0[21]$	2^4	$1 \times 2^4 \times 2^{36}$ (SB)
Total				$2^{54.15}$ (SB)

Table 6: Partial-sum technique to compute $X_0[31]$, where L_1 and $\tilde{L}_0[1, 4, 5, 10, 14, 15, 18, 19, 22, 23, 24, 25, 28, 29]$ are guessed in advance.

Step	Guessed key	Stored nibbles	(size)	Complexity (unit)
0	-	$Z_3[3, 9, 12, 16, 17, 26, 27, 30, 31], \varphi^6(T_1)[9], \texttt{mix}(X_1[24], X_1[29])$	2^{44}	
1	$\tilde{L}_0[9]$	$Z_3[3, 12, 16, 17, 26, 27, 30, 31], X_3[9], \texttt{mix}(X_1[24], X_1[29])$	2^{40}	$2^4 \times 2^{44} \times 1$ (SB)
2	$\tilde{L}_0[3]$	$Z_3[12, 16, 17, 26, 27, 30, 31], \texttt{mix}(X_3[3], X_3[9]), \texttt{mix}(X_1[24], X_1[29])$	2^{36}	$2^4 \times 2^{40} \times 2^4$ (SB)
3	$\tilde{L}_0[12]$	$Z_3[16, 17, 26, 27, 30, 31], Z_2[2], \texttt{mix}(X_1[24], X_1[29])$	2^{32}	$2^4 \times 2^{36} \times 2^8$ (SB)
4	$(\tilde{L}_1[2])$	$Z_3[16, 17, 26, 27, 30, 31], X_2[18], \texttt{mix}(X_1[24], X_1[29])$	2^{32}	$1 \times 2^{32} \times 2^{12}$ (SB)
5	$\tilde{L}_0[16]$	$Z_3[17, 26, 27, 30, 31], X_3[16], X_2[18], \texttt{mix}(X_1[24], X_1[29])$	2^{32}	$2^4 \times 2^{32} \times 2^{12}$ (SB)
6	$\tilde{L}_0[26]$	$Z_3[17, 27, 30, 31], \texttt{mix}(X_3[16], X_3[26]), X_2[18], \texttt{mix}(X_1[24], X_1[29])$	2^{28}	$2^4 \times 2^{32} \times 2^{16}$ (SB)
7	$\tilde{L}_0[31]$	$Z_3[17, 27, 30], Z_2[24], X_2[18], \texttt{mix}(X_1[24], X_1[29])$	2^{24}	$2^4 \times 2^{28} \times 2^{20}$ (SB)
8	$(\tilde{L}_1[24])$	$Z_3[17, 27, 30], \texttt{mix}(X_2[18], X_2[24]), \texttt{mix}(X_1[24], X_1[29])$	2^{20}	$1 \times 2^{24} \times 2^{24}$ (SB)
9	$\tilde{L}_0[17]$	$Z_3[27, 30], X_3[17], \texttt{mix}(X_2[18], X_2[24]), \texttt{mix}(X_1[24], X_1[29])$	2^{20}	$2^4 \times 2^{20} \times 2^{24}$ (SB)
10	$\tilde{L}_0[27]$	$Z_3[30], \texttt{mix}(X_3[17], X_3[27]), \texttt{mix}(X_2[18], X_2[24]), \texttt{mix}(X_1[24], X_1[29])$	2^{16}	$2^4 \times 2^{20} \times 2^{28}$ (SB)
11	$\tilde{L}_0[30]$	$Z_2[29], \texttt{mix}(X_2[18], X_2[24]), \texttt{mix}(X_1[24], X_1[29])$	2^{12}	$2^4 \times 2^{16} \times 2^{32}$ (SB)
12	$(\tilde{L}_1[29])$	$Z_1[23], \texttt{mix}(X_1[24], X_1[29])$	2^8	$1 \times 2^{12} \times 2^{36}$ (SB)
13	$(\tilde{L}_0[23])$	$Z_0[31]$	2^4	$1 \times 2^8 \times 2^{36}$ (SB)
14	$(\tilde{L}_1[31])$	$X_0[31]$	2^4	$1 \times 2^4 \times 2^{36}$ (SB)
Total				$2^{54.15}$ (SB)

Each list for the meet-in-the-middle contains 2^{44} key candidates. Therefore, we regard the cost of sorting and matching two lists as negligible. When we regard SB and PD as $\frac{1}{16}$RF and $4\texttt{RF} = \frac{4}{16}\texttt{ENC}$, respectively, the total time complexity is $2^{234.11}$ with $s = 6$. Thus, the data complexity is 6×2^{44}. Each partial-sum procedure has to store at most 2^{52} s-bit values. Therefore, the memory complexity is about 2^{52}.

6 Conclusion and Future Works

In this paper, we further improved the tool for finding integral distinguishers proposed in [HSE23, HGSE23]. Using this new tool, we could exploit the MixColumns property of QARMAv2 to find new integral distinguishers for QARMAv2 that are more efficient in terms of integral key recovery. Then, we leveraged the combination of meet-in-the-middle and partial-sum techniques to propose the first concrete key recovery attacks on QARMAv2. Our CP model to search for integral distinguishers is not limited to QARMAv2 and can be applied to similar designs such as MANTIS and CRAFT. In summary, we provided a 13-round attack on QARMAv2-64 ($\mathscr{T} = 1$), a 14-round attack on QARMAv2-64 ($\mathscr{T} = 2$), and a 16-round attack on QARMAv2-128 ($\mathscr{T} = 2$). Although our attacks do not threaten the security of QARMAv2 in practice, they shed more light on the security of QARMAv2 and initiate further research on the security of QARMAv2.

References

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 family of tweakable block ciphers. *IACR Transactions on Symmetric Cryptology*, 2023(3):25–73, Sep. 2023. doi:10.46586/tosc.v2023.i3.25-73.

[ADG+19] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooij, Gregor Leander, and Yosuke Todo. Zero-correlation attacks on tweakable block ciphers with linear tweakey expansion. *IACR Transactions on Symmetric Cryptology*, 2019(1):192–235, Mar. 2019. doi:10.13154/tosc.v2019.i1.192-235.

[BCG+] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer. doi:10.1007/978-3-642-34961-4_14.

[Bey23] Tim Beyne. An invariant of the round function of QARMAv2-64. Cryptology ePrint Archive, Report 2023/963, 2023. URL: https://eprint.iacr.org/2023/963.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO 2016*, pages 123–153. Springer, 2016. doi:10.1007/978-3-662-53008-5_5.

[BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019. doi:10.13154/tosc.v2019.i1.5-45.

[BLNW12] Andrey Bogdanov, Gregor Leander, Kaisa Nyberg, and Meiqin Wang. Integral and multidimensional linear distinguishers with correlation zero. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 244–261. Springer, 2012. doi:10.1007/978-3-642-34961-4_16.

[BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptogr.*, 70(3):369–383, 2014. doi:10.1007/s10623-012-9697-z.

[BW12] Andrey Bogdanov and Meiqin Wang. Zero correlation linear cryptanalysis with reduced data complexity. In *FSE 2012*, volume 7549 of *LNCS*, pages 29–48. Springer, 2012. doi:10.1007/978-3-642-34047-5_3.

[Com16] Arm Community. Armv8-a architecture: 2016 additions, 2016. URL: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/armv8-a-architecture-2016-additions.

[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.

[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.

[Gue16] Shay Gueron. A memory encryption engine suitable for general purpose processors. *Cryptology ePrint Archive, Report 2016/204*, 2016. URL: http://eprint.iacr.org/2016/204.

[HBS21] Hosein Hadipour, Nasour Bagheri, and Ling Song. Improved rectangle attacks on SKINNY and CRAFT. *IACR Trans. Symmetric Cryptol.*, 2021(2):140–198, 2021. doi:10.46586/tosc.v2021.i2.140-198.

[HGSE23] Hosein Hadipour, Simon Gerhalter, Sadegh Sadeghi, and Maria Eichlseder. Improved search for integral, impossible-differential and zero-correlation attacks: Application to ascon, forkskinny, skinny, mantis, present and qarmav2. IACR Cryptology ePrint Archive, Report 2023/1701, 2023. URL: https://eprint.iacr.org/2023/1701.

[HNE22] Hosein Hadipour, Marcel Nageler, and Maria Eichlseder. Throwing boomerangs into feistel structures: Application to CLEFIA, WARP, LBlock, LBlocks and TWINE. *IACR Trans. Symmetric Cryptol.*, 2022(3):271–302, 2022. doi:10.46586/tosc.v2022.i3.271-302.

[HSE23] Hosein Hadipour, Sadegh Sadeghi, and Maria Eichlseder. Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks. In *EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 128–157. Springer, 2023. doi:10.1007/978-3-031-30634-1_5.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014. doi:10.1007/978-3-662-45608-8_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.

[Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233, 1994. doi:10.1007/978-1-4615-2694-0_23.

[NSB+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[PF] Laurent Perron and Vincent Furnon. OR-Tools. URL: https://developers.google.com/optimization/.

[Sec17] Qualcomm Product Security. Pointer authentication on Armv8.3: Design and analysis of the new software security instructions, 2017. URL: https://www.qualcomm.com/documents/whitepaper-pointer-authentication-armv83.

[SLR+15] Bing Sun, Zhiqiang Liu, Vincent Rijmen, Ruilin Li, Lei Cheng, Qingju Wang, Hoda AlKhzaimi, and Chao Li. Links among impossible differential, integral and zero correlation linear cryptanalysis. In *CRYPTO 2015*, volume 9215 of *LNCS*, pages 95–115. Springer, 2015. doi:10.1007/978-3-662-47989-6_5.

[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.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015. doi:10.1007/978-3-662-46800-5_12.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 648–678, 2016. doi:10.1007/978-3-662-53887-6_24.

A 12-Round Integral Distinguisher for QARMAv2-128 ($\mathscr{T} = 2$)

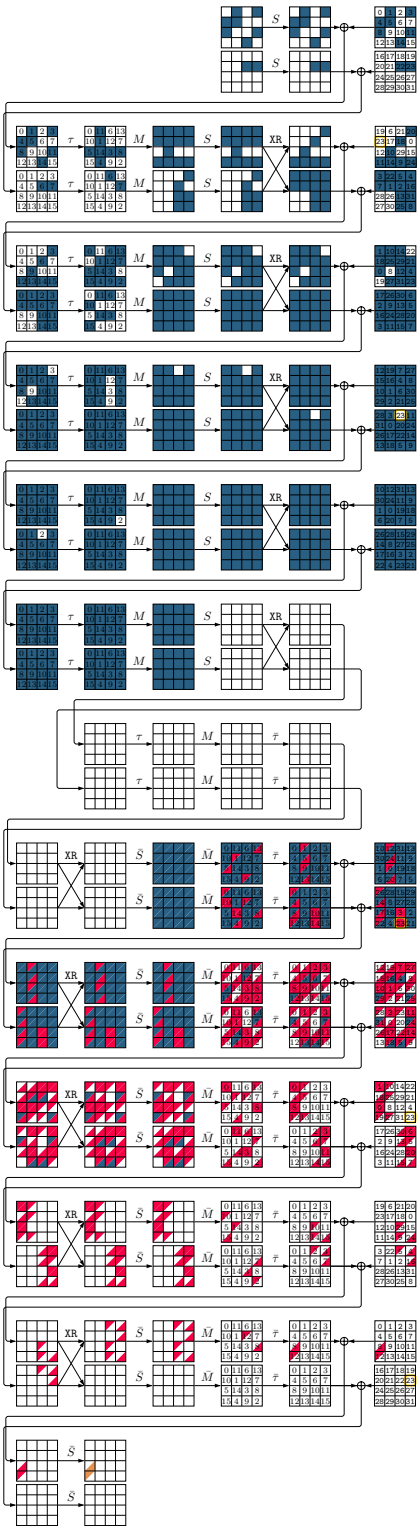


Figure 15: 12-round integral distinguisher for QARMAv2-128 ($\mathscr{T} = 2$). Data complexity 2^{96}.