Breaking the Twinkle Authenticated Encryption Scheme and Analyzing Its Underlying Permutation

Debasmita Chakraborty[1], Hosein Hadipour[2], Anup Kumar Kundu[3], Mostafizar Rahman[4], Prathamesh Ram[5], Yu Sasaki[6], Dilip Sau[7], and Aman Sinha[8]

[1] Graz University of Technology, Graz, Austria
`debasmitachakraborty1@gmail.com`,
[2] Ruhr University Bochum, Bochum, Germany
`hsn.hadipour@gmail.com`,
[3] Indian Statistical Institute, Kolkata 700108, India
`anupkundumath@gmail.com`,
[4] Kyoto University, Kyoto, Japan
`mrahman454@gmail.com`
[5] Indian Institute of Technology Bhilai, India
`rprathamesh@iitbhilai.ac.in`,
[6] NTT Social Informatics Laboratories and NIST Associate, Tokyo, Japan
`yusk.sasaki@ntt.com`
[7] Indian Institute of Technology Kharagpur, India
`dilip.sau@kgpian.iitkgp.ac.in`,
[8] Nanyang Technological University, Singapore
`aman0804@gmail.com`

Abstract. This paper studies the `Twinkle` family of low-latency symmetric key schemes designed by Wang et al. (CiC 2024). In particular, it presents cryptanalysis of both the mode and the underlying primitive. `Twinkle` is a PRF-based design, and an authenticated encryption scheme `Twinkle-AE` is specified based on a dedicated PRF called `Twinkle-PRF`. To achieve low latency, `Twinkle-PRF` uses a large key and state to produce sufficient randomness in a single step. `Twinkle-AE` uses a 1024- or 512-bit key for authentication and generates a t-bit tag, where $t \in \{64, 128\}$. It claims to provide t bits of integrity. Several `Twinkle-AE` parameter sets claim higher confidentiality than integrity. In this setup, for any ciphertext, an adversary can obtain the message after $O(2^t)$ decryption attempts by guessing the tag, allowing attacks in the chosen-ciphertext setting. We show that a 1024- or 512-bit authentication key can be recovered using only $O(2^t)$ queries. The recovered authentication key enables the generation of valid ciphertexts for arbitrary plaintexts, thus achieving universal forgery. In the second part of the paper, we perform cryptanalysis on reduced-round variants of the 1280-bit public permutation `Twinkle-P`, which serves as a core component of `Twinkle-PRF`. We investigate impossible differential, zero-correlation linear, integral, and differential-linear distinguishers by developing automated analytic tools. We provide practical distinguishers for up to 5 rounds, and the longest distinguisher reaches 6 rounds with a complexity of $2^{74.32}$. This surpasses the round bounds evaluated by the designers. We stress that our attacks on mode exploits the gap between the claimed confidentiality and integrity levels, thus have no impact on the parameter sets having the same security level. Our attacks on the permutation do not have any significant impact on the whole specifications. Moreover, we note that `Twinkle-AE-512b`/`Twinkle-AE-1024b` and `Twinkle-PA` remain secure, and the versions we attacked would also be secure if the claimed confidentiality level matched the integrity level.

Keywords: Cryptanalysis · Lightweight cryptography · Low-latency primitive · `Twinkle` · authentication · confidentiality · permutation

1 Introduction

Evolving from general-purpose schemes, symmetric-key cryptography is increasingly adopting domain-specific designs tailored for particular use cases. This shift, driven by efforts such as NIST's lightweight cryptography standardization [3] and the CAESAR competition [1], reflects diverse requirements across domains, including lightweight applications, high throughput, and strong security guarantees. A prominent domain in this context is low-latency cryptography, which plays a vital role in securing memory and system components in modern computing environments. Some key ciphers in this area include PRINCE [13], MANTIS [9], QARMA [7], SPEEDY [25], Orthros [8], Gleeok [4], SCARF [14], and BipBip [10]. With rising threats from physical and software-based memory attacks, hardware vendors now integrate cryptographic protections through solutions like Intel Software Guard Extensions (SGX) [19], AMD Secure Encrypted Virtualization (SEV) [2], and ARM CCA [5]. These mechanisms reflect varying levels of protection offered by secure memory encryption engines (MEEs), including features like ciphertext uniqueness and replay protection [17].

While encryption for memory protection has received considerable attention, the design of authenticated encryption (AE) and message authentication codes (MACs) tailored for system-level security, such as Pointer Authentication (PA), remains less explored. Existing industrial implementations (e.g., ARM's PAC [6], Intel's C^3 [26]) often rely on domain-specific MACs or tweakable block ciphers optimized for minimal latency.

Building on these developments, Wang et al. introduced the `Twinkle` framework [33] to address the need for efficient cryptographic mechanisms at the system level. This framework provides two key components: (i) a nonce-based AE scheme designed for memory encryption, and (ii) a lightweight MAC optimized for pointer authentication. Both constructions are based on a compact and novel pseudorandom function (PRF), enabling efficient hardware implementation.

In this work, we present a comprehensive security analysis of the `Twinkle` family, a set of low-latency schemes designed to provide stronger protection against replay attacks in memory encryption scenarios. The design adopts a stream cipher structure combined with a Wegman-Carter message authentication code (WC-MAC) [34], deriving both the keystream and authentication mask from a single PRF, which helps reduce overall latency. This structure enables time-aligned encryption, plaintext-independent computation, and pre-processing when the nonce is known in advance. These features further minimize delay. The 1280-bit tailored PRF state allows efficient processing of entire cache lines, while the hardware-optimized round function, which includes diffusion-friendly S-boxes and double-lane rotations, improves both performance and security. `Twinkle-PA`, a matching MAC that uses the same PRF, supports compact and efficient integration on shared hardware.

A key challenge in analyzing `Twinkle-AE` stems from its large 1280-bit internal state. This design allows a greater number of active S-boxes per round, thereby improving security against differential and linear attacks. Additionally, at least 128 bits of the state remain concealed, complicating attempts at impossible differential and guess-and-determine attacks. Furthermore, the design introduces strong diffusion early in the computation: input data is spread across the state using multiple permutation matrices. Such a strong underlying structure leads the designers to claim the security of `Twinkle-AE` based on its underlying `Twinkle-PRF` and WC-MAC. Another aspect of `Twinkle-AE` is the use of the initialization vector (IV) as an only input to its underlying `Twinkle-PRF`, which makes it considerably more difficult to mount attacks targeting the PRF.

The designers claim that the security of the `Twinkle-AE` family fundamentally relies on the robustness of its underlying PRF, which is based on a large-state Even-Mansour construction. Assuming a random permutation and distinct key components, they argue that the construction achieves a security bound of $O(2^{640})$ for Twinkle's 1280-bit state, which is well beyond its intended security targets. The `Twinkle-PRF` has undergone extensive analysis against a wide range of classical cryptanalytic techniques, including differential, linear, integral, impossible differential, and meet-in-the-middle attacks. Based on these properties, the designers assert that the PRF is secure for use in both memory encryption and pointer authentication scenarios, under the assumption that the key is securely stored within the processor and not subject to related-key attacks.

The designers of `Twinkle` claim security based on the strength of the `Twinkle-PRF` and the WC-MAC components. Each part of `Twinkle-AE`, namely the encryption and MAC mechanisms, individually satisfies standard security guarantees. However, our analysis shows that the universal hash function introduces a weakness that an attacker can exploit when the number of queries can exceed the output size. Hence, as our first objective, we identify vulnerabilities that do not require directly targeting the underlying PRF.

In addition, although the designers have explored the security of the underlying permutation against cryptanalytic attacks, they have rarely presented concrete distinguishers. In particular, they have not examined the security of the permutation against differential-linear attacks, which often serve as some of the most powerful distinguishers for cryptographic permutations. To address this gap, we construct concrete differential-linear distinguishers and either significantly strengthen the existing analysis or provide the first known results in this direction.

1.1 Our Contributions

We provide a focused cryptanalytic evaluation of the `Twinkle-AE` scheme, addressing gaps in the analysis of the `Twinkle-P` permutation and key-recovery resistance. Our contributions are twofold:

1. **Authentication key-recovery attacks:** For AE, even if a nonce and a ciphertext are known, a corresponding M is not accessible without knowing a tag T. However as observed by Hosoyamada et al. [24] against Rocca [28], higher confidentiality than integrity allows an adversary to exhaustively guess T with 2^t queries, allowing M to be leaked to the adversary. By combining this idea with the structure of `Twinkle-AE`, we demonstrate authentication key-recovery attacks on `Twinkle-AE` for the versions claiming higher confidentiality than integrity, which allows an adversary to recover a 1024- or 512-bit authentication key K' with $O(2^t)$ queries in the nonce-respecting setting, where $t \in \{64, 128\}$. The recovered K' enables the generation of valid ciphertexts for arbitrary plaintexts, thus achieving universal forgery, which breaks the notion of indistinguishability under chosen-ciphertext attack (IND-CCA). Importantly, these attacks do not exploit weaknesses in the underlying `Twinkle-PRF`, but rather in the use of the universal hash function within the WC-MAC construction. Additionally, we present a variant of the attack that succeeds with only $O(1)$ queries in the nonce-misuse setting. Note that because the IV size is 128 bits, repeat of the same nonce is inevitable after 2^{128} queries for `Twinkle-AE-512c` and `Twinkle-AE-1024c` that claim 256-bit confidentiality even with the restriction that the attacker can only make queries in the chosen-plaintext setting. We also show that `Twinkle-AE` does not provide key-committing security, allowing an adversary to create distinct keys that produce the same ciphertext with $O(1)$ cost.
2. **Analysis of the underlying permutation:** We analyze the `Twinkle-P` permutation, the core component of `Twinkle-AE`, against several cryptanalytic distinguishers. Table 1 summarizes our findings. Notably, we present practical differential-linear (DL) distinguishers for up to 5 rounds of `Twinkle-P`, as well as a strong theoretical DL distinguisher for 6 rounds. For example, the data complexity of our 5-round differential-linear distinguisher is $2^{5.70}$ randomly chosen input pairs, or equivalently, $2 \cdot 2^{5.70}$ chosen inputs. While the designers claimed that no 7-round impossible differential trail could be constructed even with full adversarial control over all 1280 bits, we match this bound by explicitly constructing a 6-round impossible differential attack. Moreover, we provide zero-correlation (ZC) distinguishers for up to 6 rounds. We also leverage the connection between ZC and integral distinguishers, along with the constraint programming (CP) based search methods proposed in [23,21], to identify ZC-based integral attacks. Additionally, we use the bit-based division property to derive practical integral

Distinguisher	#Rounds	#Distinguishers	Attack complexity	Ref.
Differential	4	–	$> 2^{58}$	[33]
Linear	4	1	2^{60}	[33]
Truncated Differential	3.5	1	$2^{7.4}$	[33]
Differential-Linear	**4**	80	2	subsection 5.6
	5	80	$2 \cdot \mathbf{2^{5.70}}$	subsection 5.6
	6	80	$2 \cdot \mathbf{2^{73.32}}$	subsection 5.6
Impossible Differential	4	$80 \cdot 2^{1820}$	–	subsection 5.3
	5	$80 \cdot 2^{1148}$	–	subsection 5.3
	6	$80 \cdot 2^{356}$	–	subsection 5.3
Zero-Correlation Linear	4	$80 \cdot 2^{1278}$	–	subsection 5.4
	5	$80 \cdot 2^{1140}$	–	subsection 5.4
	6	$80 \cdot 2^{16}$	–	subsection 5.4
Integral	3	80	2	subsection 5.4
	4	80	2^{4}	subsection 5.4
	5	80	2^{12}	subsection 5.5

Table 1: Summary of distinguishers for `Twinkle-P`

distinguishers for up to 5 rounds of `Twinkle-P`. We provide the source code of our tools as well as the experimental verifications at the following link: `https://github.com/hadipourh/twinkle`.

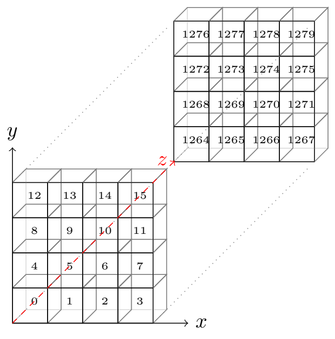
1.2 Paper Organization

The rest of the paper is organized as follows. Section 2 introduces the notation used throughout the paper. Section 3 describes the specification of `Twinkle`. Section 4 presents our attacks against `Twinkle-AE`. Section 5 provides the cryptanalysis of reduced rounds of `Twinkle-P`. Finally, Section 6 concludes the paper.

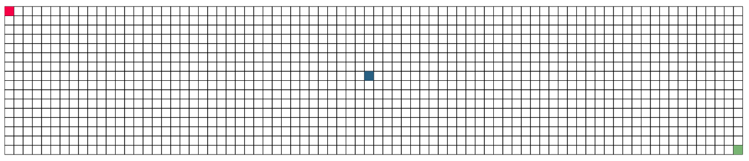
2 Notation

Let $\mathbb{Z}_m$ denote the set $\{0, 1, \ldots, m-1\}$ for a positive integer m. Assume that $S = S[m-1] \parallel \cdots \parallel S[1] \parallel S[0]$ is an m-bit string, where $S[0]$ (resp. $S[m-1]$) represents the least significant bit (LSB) (resp. most significant bit (MSB)). We assume that m is a multiple of 16, that is, $m = 16 \times \ell$ for some $\ell \in \mathbb{Z}_n$. According to the `Twinkle-AE` specification [33], the internal state of `Twinkle-P` is arranged into a $4 \times 4 \times \ell$ three-dimensional array, as illustrated in Figure 1a. We use (x, y, z) or, alternatively, $S[x][y][z]$ to indicate the position of each bit within the 3D representation, where $x, y \in \{0, \ldots, 3\}$ and $0 \leq z \leq \ell - 1$. The entry $S[x][y][z]$ within the 3D representation corresponds to $S[x + 4 \cdot y + 16 \cdot z]$ within the one-dimensional (1D) representation. Similar to the specification [33], we use $\bullet$ to denote that one coordinate can take all possible values. For example, for a given pair (y, z), $S[\bullet][y][z]$ represents $S[x][y][z]$ for all $x \in \{0, \ldots, 3\}$. We refer to $S[\bullet][y][z]$, $S[x][\bullet][z]$, $S[x][y][\bullet]$, and $S[\bullet][\bullet][z]$ as *row*, *column*, *lane*, and *slice*, respectively.

Besides the 3D representation, we can also represent the internal state as a two-dimensional $16 \times \ell$ array, as shown in Figure 1b. In this case, we assign $S[x][y][z]$ from the 3D representation to $S[i][j]$ in the 2D representation, where $j = z$ and $i = x + 4 \cdot y$. Therefore, the jth slice in the 3D representation corresponds to the jth column of the 2D representation, where each column is filled from top to bottom. That is, the topmost bit of each column in 2D representation is the first bit of each slice. Additionally, each row of the 2D representation corresponds to a lane of the 3D representation. For example, the cells marked by 🟥, 🟦, and 🟩 in Figure 1b correspond to $S[0][0][0]$, $S[3][1][39]$, and $S[3][3][79]$ in the 3D representation, respectively. We recall that $S[0][0][0]$ is the LSB of the m-bit string S, and the leftmost bit in each row of the 2D representation represents the LSB of the corresponding lane. We denote left (i.e. from LSB to MSB) and right (i.e. from MSB to LSB) bit rotations within an n-bit string by $\lll_n$ and $\ggg_n$, respectively.



(a) 3D representation of a the `Twinkle` internal state



(b) 2D representation of the `Twinkle` internal state

Fig. 1: Different views of the `Twinkle` state

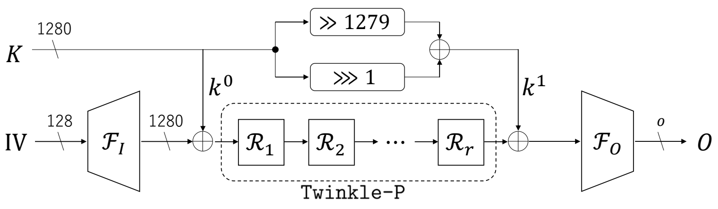


Fig. 2: Overview of `Twinkle-PRF`

3 Specification of `Twinkle`

`Twinkle` presents two modes of operations for a PRF; an AE mode `Twinkle-AE` and a MAC mode specialized for pointer authentication `Twinkle-PA`. `Twinkle` defines a specific instance by designing a dedicated PRF `Twinkle-PRF`, which is computed with a dedicated public permutation `Twinkle-P`.

3.1 `Twinkle-PRF`

`Twinkle-PRF` : $\{0,1\}^{1280} \times \{0,1\}^{128} \mapsto \{0,1\}^o$ is a function that takes a 1280-bit key K and 128-bit IV as input and returns an o-bit string where $o \in \{1,\dots,1152\}$. When a key is fixed, which is a more relevant scenario for attacking specific users, `Twinkle-PRF` is a 128-bit to o-bit function.

Overall, `Twinkle-PRF` performs the following three operations. First, an input expansion function $\mathcal{F}_I : \{0,1\}^{128} \mapsto \{0,1\}^{1280}$ is computed to create a 1280-bit state S from the 128-bit IV. Then, the Even-Mansour construction is computed with a 1280-bit key K and `Twinkle-P`. Finally, an output compression function $\mathcal{F}_O : \{0,1\}^{1280} \mapsto \{0,1\}^o$ is computed to produce an o-bit function. The overall structure of `Twinkle-PRF` is depicted in Figure 2

$\mathcal{F}_I$ duplicates the 128-bit IV to make 10 copies of IV, each is applied a different bit-permutation σ_i. Namely, a 1280-bit state S is computed by $S \leftarrow \sigma_9(\mathrm{IV})\|\sigma_8(\mathrm{IV})\|\cdots\|\sigma_0(\mathrm{IV})$, where each σ_i moves the bit position j of IV to the bit position $a_i \cdot j + b_i \bmod 128$ for $j \in \{0,\dots,127\}$, and the values of a_i and b_i are defined in Table 2.

i	0	1	2	3	4	5	6	7	8	9
a_i	1	3	5	7	11	13	17	19	23	29
b_i	0	1	2	3	4	5	6	7	8	9

Table 2: Parameters of Bit Permutation σ_i.

$\mathcal{F}_O$ generates an o-bit string by computing $\mathrm{Trunc}_o(S \oplus (S \gg 128))$, where Trunc_n represents the least significant n bits of a bit string.

In the Even-Mansour construction in the middle, the 1280-bit state is updated by $S \leftarrow \texttt{Twinkle-P}(S \oplus k^0) \oplus k^1$, where k^0 is the master key K, `Twinkle-P` is a 1280-bit public permutation specified later, and k^1 is another 1280-bit key computed by $k^1 \leftarrow (K \ggg 1) \oplus (K \gg 1279)$.

Note that the number of rounds inside `Twinkle-P` depends on the security level required by an upper-layer protocol, which is denoted by m, $m \in \{64, 128, 256\}$. This indicates that `Twinkle-PRF` also depends on m. `Twinkle-PRF` is also parameterized by an output length o. To clarify those parameters, the designers introduced a notion of $\texttt{Twinkle}_o^m$ to denote the `Twinkle-PRF` with a security parameter of m and the output length of o.

3.2 `Twinkle-AE`

`Twinkle-AE` is an AE scheme specialized for encrypting the cache line, therefore, the plaintext size, denoted by c, is limited to two choices, either 512 bits or 1024 bits, i.e. $c \in \{512, 1024\}$, and the corresponding scheme is called `Twinkle-AE-512` and `Twinkle-AE-1024`, respectively. IV serves as a unique nonce, which is never used more than once under the same key. Unlike standard AEAD schemes, `Twinkle-AE` does not take associated data as input. The ciphertext size is the same as the plaintext size, and it generates a t-bit tag for the message authentication, where $t \in \{64, 128\}$. Besides, `Twinkle-AE` supports two levels of confidentiality; 128 bits and 256 bits, in which the choice of the confidentiality level only affects the number of rounds inside `Twinkle-P`, and makes no difference as long as `Twinkle-PRF` is viewed as a block-box. `Twinkle-AE` specifies six valid combinations of c, t, and the confidentiality level as listed in Table 3.

Versions	Confidentiality	Integrity (t)
`Twinkle-AE-512a`	128	64
`Twinkle-AE-512b`	128	128
`Twinkle-AE-512c`	256	128
`Twinkle-AE-1024a`	128	64
`Twinkle-AE-1024b`	128	128
`Twinkle-AE-1024c`	256	128

Table 3: `Twinkle-AE` Versions and Security in Bits

Encryption. The diagram of the encryption procedure is depicted in Figure 3. Encryption of `Twinkle-AE` takes as input a 1024-bit encryption key K, a c-bit authentication key K', a c-bit message M, and a 128-bit IV. It outputs a c-bit ciphertext C and t-bit tag T.

Encryption of `Twinkle-AE` first computes `Twinkle-PRF` to generate a $t + c$ bit string denoted by $O_t \| O_c$, from a 128-bit IV and a 1280-bit key K. Namely,

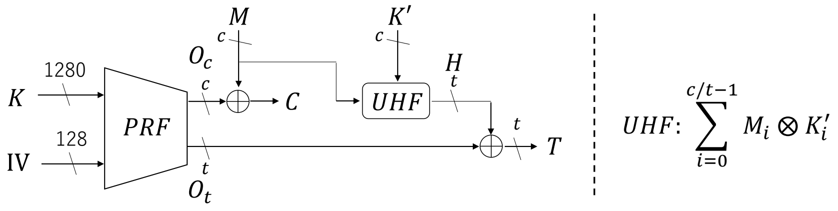


Fig. 3: Encryption Diagram of `Twinkle-AE`

$O_t \| O_c \leftarrow$ $\texttt{Twinkle}^m_{t+c}(\mathrm{IV}, K)$, where m is either 128 or 256 depending on the version of `Twinkle-AE`. O_c, c bits of the generated string, is used as a key stream. Namely, O_c is XORed to M, and the result is output as a c-bit ciphertext C.

O_t, t bits of the generated string, is used to generate a t-bit tag T in the WC-MAC paradigm. In details, M and K' are first divided into t-bits of strings $M_{c/t-1}\| \cdots \|M_1\|M_0 \leftarrow M$ and $K'_{c/t-1}\| \cdots \|K'_1\|K'_0 \leftarrow K'$, then a universal hash function $H = \sum_{i=0}^{c/t-1} M_i \otimes K'_i$ is computed, wherein $\otimes$ represents the multiplication in the finite field F_2^t, and finally $O_t \oplus H$ is output as a tag T.

3.3 `Twinkle-PA`

`Twinkle-PA` is a MAC scheme designed for a pointer authentication, namely to authenticate a 64-bit pointer address PT and a 64-bit context CT. With a 1280-bit key K, it generates a t-bit tag T, where $1 \le t \le 32$.

`Twinkle-PA` consists of a single call of `Twinkle-PRF`. $CT\|PT$ forms a 128-bit IV, and the output of `Twinkle-PRF` is directly used as T. `Twinkle-PA` provides 64-bit security against offline attacks. Namely, $T \leftarrow \texttt{Twinkle}^{64}_t(CT\|PT, K)$.

3.4 `Twinkle-P`

Each round ($\mathcal{R}$) in `Twinkle-P` consists of five operations, namely `S-box` (SB), $\texttt{LaneRotation}_0$ (LR_0), `MixSlice` (MS), $\texttt{LaneRotation}_1$ (LR_1), and `AddConstant` (AC):

$$\mathcal{R} = AC \circ LR_1 \circ MS \circ LR_0 \circ SB.$$

`S-box` (SB): This step applies a 4-bit S-box S to each row of the internal state:

$$S[0][y][z] \parallel \cdots \parallel S[3][y][z] \leftarrow \texttt{S-box}(S[0][y][z] \parallel \cdots \parallel S[3][y][z]),$$

where $y \in \mathbb{Z}_4$ and $z \in \mathbb{Z}_{80}$. Table 4 shows the lookup table for `Twinkle`'s S-box.

$\texttt{LaneRotation}_0$ (LR_0): This step applies a right rotation using the offsets O_0 (defined in Table 5) to each lane of the state:

$$S[x][y][\cdot] \leftarrow S[x][y][\cdot] \ggg_{80} (O_0[x+4y] \bmod 80), \quad \text{for all } x, y \in \mathbb{Z}_4.$$

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

Table 4: `Twinkle`'s S-box

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O_0	20	24	38	77	49	66	30	40	76	15	46	50	17	18	61	62
O_1	63	45	34	39	32	43	60	66	54	26	55	36	61	12	15	35

Table 5: Offsets for `LaneRotation`

`MixSlice` (MS): This step applies a linear transformation to each slice. For all $z \in \mathbb{Z}_{80}$, we compute:

$$S[\cdot][\cdot][z] \leftarrow S[\cdot][\cdot][z] \oplus (S[\cdot][\cdot][z] \lll_{16} 5) \oplus (S[\cdot][\cdot][z] \lll_{16} 12)\,.$$

`LaneRotation`$_1$ (LR_1): This operation is similar to LR_0 but uses the offset values O_1 from Table 5.

`AddConstant` (AC): At each round i, this step XORs a fixed 1280-bit round constant RC_i with the state.

As mentioned in Section 3.1, the number of rounds in `Twinkle-P` depends on the security level m. Specifically, for $m = 64$, 128, and 256, the total number of rounds is 5, 9.5, and 18.5, respectively. Here, “0.5 round” refers to an operation consisting of the application of `S-box` followed by a `LaneRotation`$_0$ on the 1280-bit state.

4 Generic Authentication Key Recovery for `Twinkle-AE`

By observing that some `Twinkle-AE` versions claim higher confidentiality than integrity, in subsection 4.1, we present authentication key-recovery attacks on `Twinkle-AE` mode, which recovers a c-bit key K' with $O(2^t)$ queries in the nonce-respecting setting. This eventually allows us, for an arbitrary choice of a message, to generate a ciphertext that passes the verification and is decrypted to the chosen message. This means that the generated plaintext by the decryption of `Twinkle-AE` is predictable, or more precisely, controllable in the chosen-ciphetext setting. Hence, our attacks break the IND-CCA security when confidentiality is higher than integrity, namely for `Twinkle-AE-512a`, `Twinkle-AE-512c`, `Twinkle-AE-1024a`, and `Twinkle-AE-1024c`. We then present variants of this attack that require only $O(1)$ queries in the nonce-misuse setting in subsection 4.2. Note that the attack does not exploit any property of `Twinkle-PRF`, but exploits the property of the universal hash function.

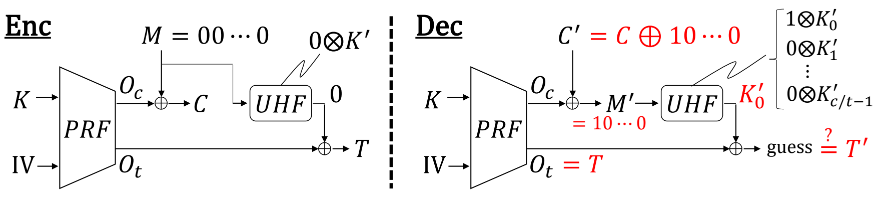


Fig. 4: Diagram of Nonce-Respecting Authentication Key Recovery

4.1 Nonce-Respecting Attacks with $O(2^t)$ Queries

Observations on Higher Confidentiality than Integrity. First, we observe that each of the encryption part and the MAC part of `Twinkle-AE` is rather solid. The ciphertext is generated by XORing the message with a PRF output, where a nonce IV is involved as a PRF input. This is a secure construction, identical to one-time pad, as long as the PRF output cannot be distinguished from a random string. The tag generation part is the simplest instantiation of the WC-MAC, where the message processed by a universal hash function is XORed with the PRF output. Hence, as long as the security of the encryption part and the MAC part are identical, the construction looks solid.

However, several `Twinkle-AE` versions are designed to provide higher confidentiality than integrity. This drastically changes the situation because attackers can make queries beyond the limit imposed by integrity, i.e. more than 2^t queries. As observed by Hosoyamada et al. [24], chosen-ciphertext attacks could be critical for confidentiality in this case. This contrasts with typical AE security, where such attacks usually have negligible impact because forgery attempts do not succeed below the integrity limit, and no plaintext is revealed from decryption.

Recovering K'. With the above observation, we show that the authentication key K' can be efficiently recovered. The attack is depicted in Figure 4.

The universal hash function (UHF) is a sum of field multiplications of M and K'. When M is set to 0, the output of UHF is $0 \otimes K' = 0$. Hence, when an encryption query $(\mathrm{IV}, 0)$ results in (C, T), T equals to the t-bit of the PRF output O_t. Similarly, since $M = 0$, C equals to the c-bit of the PRF output O_c.

We then play with the decryption oracle under the same IV. Let $C_0, \ldots, C_{c/t-1}$ denote the ciphertext C in every t bits. We then modify C to C' so that $C'_0 = C_0 \oplus 1$ and $C'_i = C_i$ for $i = 1, \ldots, c/t-1$. This makes the value of $M' = 1\|0\|\cdots\|0$, and the output of UHF is $(1 \otimes K'_0) \oplus \sum_{i=1}^{c/t-1}(0 \otimes K'_i) = K'_0$. The challenge here is that an attacker does not know the new tag. However, as observed in above, the attacker can make 2^t decryption queries to exhaustively try all the tags, and obtain the corresponding tag T' after 2^t queries. Then, K'_0 is immediately recovered by $K'_0 = O_t \oplus T'$.

Other K'_i can be recovered in the same procedure by swapping t-bit block in C' to i-th block. Hence, the entire K' is recovered with $c/t \cdot 2^t$ queries. Recall that the possible parameter choices of c and t are $c \in \{512, 1024\}$ and $t \in \{64, 128\}$, so c/t is a small constant 4, 8, or 16.

A pseudo-code of the authentication key-recovery attack is given in Alg. 1.

Algorithm 1 Nonce-Respecting Authentication Key Recovery

```
Require: Accesses to decryption and encryption oracles with $K, K'$
Ensure: $K'$
 1: Set IV to an arbitrary chosen value, and set $M$ to 0.
 2: Send $(\mathrm{IV}, M)$ to an encryption oracle to get corresponding $(C, T)$.
 3: Record $O_c = C$ and $O_t = T$.
 4: for $i = 0, \ldots, c/t - 1$ do
 5:     Set $C' = C$, then $C'_i \leftarrow C_i \oplus 1$.
 6:     for $T' = 0, \ldots, 2^t - 1$ do
 7:         Send $(\mathrm{IV}, C', T')$ to decryption oracle.
 8:         if The decryption oracle returns a message $M'$ then
 9:             $K'_i \leftarrow O_t \oplus T'$.
10:         end if
11:     end for
12: end for
13: return $K'$.
```

Universal Forgery with the knowledge of K'. After the authentication key K' is recovered, along with a tuple of related values IV, O_t, and O_c, universal forgery can be performed with a negligible cost, i.e. for an arbitrary chosen value of the message M^*, the attacker can generate a ciphertext (IV, C^*, T^*) that passes the verification and is decrypted to M^*. The attack procedure is straightforward. With O_c, C^* can be computed by $C^* \leftarrow O_c \oplus M^*$. With O_t and K', T^* can be computed by $T^* \leftarrow O_t \oplus \mathrm{UHF}_{K'}(M^*)$. Note that those can be computed offline, hence no additional query is required. A pseudo-code of universal forgery is given in Alg. 2.

In addition, with the knowledge of K', universal forgery for any other IV is possible only with 1 additional known plaintext query. Given an encryption result $\tilde{C}, \tilde{T}$ for any $\tilde{\mathrm{IV}}, \tilde{M}$, the attacker can compute the corresponding $\tilde{O}_c$ and $\tilde{O}_t$ with K', and the rest of the attack is trivial.

Algorithm 2 Nonce-Respecting Universal Forgery

```
Require: $K'$, IV, $O_c$, $O_t$, and a target message $M^*$ that can be chosen arbitrary
Ensure: $(\mathrm{IV}, C^*, T^*)$ that is decrypted to $M^*$
 1: Compute $C^* \leftarrow O_c \oplus M^*$ offline.
 2: Compute $T^* \leftarrow O_t \oplus \mathrm{UHF}_{K'}(M^*)$ offline.
 3: return $(\mathrm{IV}, C^*, T^*)$.
```

4.2 Nonce-Misuse Attack with $O(1)$ Queries

Due to the nature of the key-stream XOR in encryption and WC-MAC in authentication, it is natural that both confidentiality and integrity can be broken efficiently if the same IV is iterated more than once. However, it does not necessarily mean that authentication-key recovery or universal forgery is performed efficiently. Here we show that those strong attacks are possible with $O(1)$ queries in the nonce-misuse setting.

Note that because the size of IV is 128 bits, repeat of the same nonce is inevitable after 2^{128} queries for `Twinkle-AE-512c` and `Twinkle-AE-1024c` that claim 256-bit confidentiality with a 128-bit tag even with the restriction that the attacker can only make queries in the chosen-plaintext setting.

In the nonce-misuse setting, the procedure to recover K'_0 is as follows.

1. Make an encryption query of $(\mathrm{IV}, 0\|0\|\cdots\|0)$ to get (C_0, T_0).
2. Make an encryption query of $(\mathrm{IV}, 1\|0\|\cdots\|0)$ to get (C'_0, T'_0).
3. $K'_0 = T_0 \oplus T'_0$.

Other K'_i can be recovered similarly. We omit the details to avoid redundancy.

4.3 Remarks on Key-Committing Security

The key-committing security of symmetric-key cryptography was initiated by Farshim et al. [18], which was later generalized as CMT-1 security by Bellare and Hoang [11]. Its goal is to find a ciphetext (C, T) that can be decrypted with two distinct keys, i.e. to find (K, K', IV, M) and $(\tilde{K}, \tilde{K}', \tilde{\mathrm{IV}}, \tilde{M})$ with $K \neq \tilde{K}$ and $K' \neq \tilde{K}'$ such that the corresponding output of encryption queries will collide. It has been known that the lack of key-committing security may cause attacks in several real-world use-cases e.g. braking message franking [16] or accelerating password brute-force attacks [27].

The key-committing security of `Twinkle-AE` can be broken in the same way as the key-committing attack against `GCM` [16]. The attacker first randomly chooses $K, \tilde{K}, K', \tilde{K}'$, and $\mathrm{IV} = \tilde{\mathrm{IV}}$. Then, O_c, O_t, and $\tilde{O}_c, \tilde{O}_t$ can be computed by `Twinkle-PRF`.

Let ℓ be c/t, and O_c is separated into t-bit chunks denoted by $O_{c,0}, \ldots, O_{c,\ell-1}$, and the same is applied to $\tilde{O}_c$. Then the equations to compute T and $\tilde{T}$ can be written as follows.

$$T = (O_{c,0} \oplus C_0) \cdot K'_0 \oplus \cdots \oplus (O_{c,\ell-1} \oplus C_{\ell-1}) \cdot K'_{\ell-1} \oplus O_t,$$
$$\tilde{T} = (\tilde{O}_{c,0} \oplus C_0) \cdot \tilde{K}'_0 \oplus \cdots \oplus (\tilde{O}_{c,\ell-1} \oplus C_{\ell-1}) \cdot \tilde{K}'_{\ell-1} \oplus \tilde{O}_t.$$

The goal is to choose the value of C so that $T = \tilde{T}$ is satisfied. The attacker now fixes the values of $C_1, \ldots, C_{\ell-1}$ to arbitrary chosen values. Then, all the variables but C_0 become fixed constants. Denoting the sum of all the variables not including C_0 as X and $\tilde{X}$, the equation becomes

$$X \oplus C_0 \cdot K'_0 = \tilde{X} \oplus C_0 \cdot \tilde{K}'_0.$$

This is a linear equation with 1 free variable C_0 and the value of C_0 can be obtained immediately.

5 Analysis of the Underlying Primitive

The underlying permutation used in constructing AE schemes should resist most well-known cryptanalytic distinguishers, such as impossible differential, zero-correlation, integral, and differential-linear distinguishers. Here, we analyze the security of `Twinkle-P` as a standalone permutation, regardless of how it is used in the mode of operation.

5.1 Modeling the `MixSlice` Operation

To simplify the modeling of differential, linear, and integral property propagation through `MixSlice`, we derive its matrix representation. Let us represent each 16-bit slice $S[\bullet][\bullet][z]$ as a 16-bit vector $V \in \mathbb{F}_2^{16}$, such that $S[x][y][z] = V[x + 4 \cdot y]$ for all $x, y \in \mathbb{Z}_4$. Then, the `MixSlice` operation acts as a matrix multiplication $A \times V$, where A is a 16×16 circulant matrix. The ith rows of A and A^{-1} are as follows:

$$\begin{aligned} A[i] &= \begin{bmatrix} 1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0 \end{bmatrix} \ggg i, \\ A^{-1}[i] &= \begin{bmatrix} 1\,0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,1\,1 \end{bmatrix} \ggg i, \end{aligned}$$

for $i \in \mathbb{Z}_{16}$. We use A and A^{-1} to model the propagation of differences in the forward and backward directions, respectively.

To model the propagation of linear masks in the forward and backward directions, we use $\left(A^{-1}\right)^t$ and A^t, respectively. These are also circulant 16×16 matrices, and their ith rows are as follows:

$$\begin{aligned} \left(A^{-1}\right)^t[i] &= \begin{bmatrix} 1\,1\,1\,0\,1\,0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0 \end{bmatrix} \ggg i, \\ A^t[i] &= \begin{bmatrix} 1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \end{bmatrix} \ggg i. \end{aligned}$$

As can be seen, the diffusion of differences in the forward direction (resp. linear masks in the backward direction) is lower than that in the backward direction (resp. forward direction).

5.2 Modeling the S-box

To model the propagation of differential, linear, and integral properties through `S-box`, we use the S-box Analyzer [22,21][9]. We model the S-box in three different ways: probabilistic with probabilities encoded, probabilistic without probabilities encoded, and deterministic. We use the probabilistic model with probabilities encoded to find regular differential and linear trails. We use the probabilistic model without probabilities encoded to model only the possibility of a differential or linear transition, regardless of the weight or probability of the transition. We use the deterministic model to represent differential and linear transitions with probability one. Such models are used to find impossible differential, zero-correlation, and differential-linear distinguishers. For more details on modeling the S-box using the S-box Analyzer, refer to Section 7.

[9] `https://github.com/hadipourh/sboxanalyzer`

5.3 Impossible Differential Distinguishers

The designers of `Twinkle-AE` provided some bounds for impossible differential (ID) distinguishers by analyzing the full diffusion of `Twinkle-P` in both forward and backward directions [33]. However, they did not present any concrete ID distinguisher for `Twinkle-P`. Here, we fill this gap by providing concrete ID distinguishers.

We use two different techniques to find ID distinguishers: the negative CP (MILP/SAT)[10] model proposed in [29,15], and the positive CP (MILP/SAT) model proposed in [23,21]. In the negative model, we model the probabilistic propagation of differences (without encoding probabilities) and fix both the input and output differences. We then use a CP/MILP/SAT solver to check the satisfiability of the model. If the model is unsatisfiable, the chosen input and output differences yield an impossible differential.

In the positive model, we model the deterministic propagation of differences in both forward and backward directions. In this model, we encode the difference value of each bit using three symbols: 0, 1, and ?, where ? denotes an arbitrary difference or, equivalently, a free bit with respect to the difference value. We also add extra constraints to ensure that the forward and backward propagations contradict in at least one bit along the distinguisher. As a result, any feasible solution of the positive model corresponds to an ID distinguisher. For more details on how to construct the negative and positive models, we refer to [29,15] and [23,21], respectively.

Both negative and positive CP/MILP/SAT models can find ID distinguishers for up to 6 rounds of `Twinkle-P`. An advantage of the positive model is that we do not need to fix the input and output differences. Instead, we can define an objective function that maximizes the number of bits with arbitrary difference values at the input and output when solving a single instance of the model. In this way, we can find a truncated ID with more free bits at the input and output differences (i.e., a larger cluster of ID distinguishers), by solving only one instance of the positive model.

We present the distinguishers found using the positive model in Figure 5a, Figure 5b, and Figure 5c. In these figures, the values "1" and "?" in the forward (resp. backward) difference propagation are represented using ◩ and ◸ (resp. ◿ and ◪). For example, ■ indicates that the corresponding bit takes an arbitrary difference in the forward propagation, while its difference is fixed and equal to "1" in the backward propagation. We represent zero differences using white (or no color). As a result, the presence of either ◩ or ◪ within a shape indicates a contradiction between the forward and backward propagations.

Note that the number of free bits at the input and output of the distinguisher in Figure 11a (marked by ◸ and ◿, respectively) are 824 and 996, respectively. As a result, this figure represents $824 + 996 = 1820$ impossible differential (ID) distinguishers. Based on a similar argument, Figure 5b and Figure 5c illustrate

[10] CP stands for Constraint Programming, MILP stands for Mixed Integer Linear Programming, and SAT stands for the Satisfiability Problem. Here, we consider MILP and SAT as special cases of CP.

2^{1148} and 2^{356} ID distinguishers for 5 and 6 rounds of `Twinkle-P`, respectively. Moreover, since `Twinkle-P` remains invariant under rotation along the z-axis with respect to differential and linear trails, each ID distinguisher can be duplicated into 80 distinct ones. We recall that the distinguishers discovered using the positive CP model were obtained from a single execution of the model. If we run the tool multiple times, it may yield additional solutions (i.e., distinguishers).

5.4 Zero-Correlation and ZC-based Integral Distinguishers

We note that the designers of `Twinkle` neither provided any concrete zero-correlation (ZC) distinguisher nor leveraged the connection between ZC and integral distinguishers to construct integral distinguishers. Here, we fill this gap. Analogous to the approach used for impossible differential distinguishers, we search for ZC distinguishers using both positive and negative models. Both models are capable of identifying distinguishers up to 6 rounds. However, to obtain ZC distinguishers with a higher number of free input bits, we prefer the positive model. To this end, we configure the objective function of the positive model to maximize the number of free bits in the input linear mask. Finally, we apply Theorem 1 to transform our ZC distinguishers into ZC-based integral distinguishers.

Theorem 1 (Sun et al. [30]). *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, having more free bits at the input linear mask of a ZC distinguisher (hull) results in fewer active bits in its corresponding integral distinguisher. Therefore, by maximizing the number of free bits at the input linear mask in the ZC distinguisher, we can obtain integral distinguishers with lower data complexity.

Similar to the representation used for ID distinguishers, we represent the values “1” and “?” in the forward (resp. backward) propagation of linear masks using ◩ and ◩ (resp. ◪ and ◪). Figure 7a, Figure 7b, and Figure 7c illustrate 2^{1278}, 2^{1140}, and 2^{16} ZC distinguishers that we obtained using the positive model for 4, 5, and 6 rounds of `Twinkle-P`, respectively.

We first applied our tool to 3 rounds of `Twinkle-P`. As shown in Figure 6 we discovered a zero-correlation (ZC) linear hull whose input linear mask has only one inactive bit, while the remaining 1279 input bits take a free linear mask (marked by ◩). As a result, its corresponding integral distinguisher has a data complexity of 2, where the input set assigns a fixed but random value to bits with free input linear mask (marked by ◩) and takes all possible values over the single bit with inactive linear mask.

As another interesting result, the input linear mask of our 4-round ZC distinguisher contains 1276 free bits (marked by ◩), while the remaining 4 bits are fixed to zero. Consequently, the corresponding 4-round integral distinguisher requires a data complexity of 2^4. In this distinguisher, the input set assigns fixed but random values to all free bits in terms of linear mask (marked by 🟩), while the remaining 4 bits take all possible combinations (see the input linear mask in Figure 7a). At the output, the XOR of the active cells (marked by 🟥) satisfies a zero-sum property. We experimentally verified the validity of this 4-round integral distinguisher using at least 100 randomly chosen input sets that satisfy the required input structure, and in all cases, the distinguisher held as expected.

Our 5-round ZC distinguisher, shown in Figure 7b, has an input linear mask with 1140 free bits (marked by 🟩), while the remaining 140 bits in the mask are fixed to zero. As a result, the data complexity of the corresponding integral distinguisher is 2^{140}. Additionally, since `Twinkle-P` preserves zero-correlation linear hulls under rotation along the z-axis, we can derive 80 distinct ZC distinguishers from each initial one.

5.5 Division Property Based Integral Distinguishers

Here, we use the division property [31] and construct a MILP-based model [32,35] to identify integral distinguishers for `Twinkle-P`. We found a 4-round integral distinguisher for `Twinkle-P`, which originates from an input state featuring four active bits. As a result, the data complexity of the 4-round integral distinguisher is 2^4. Let S^i denote the i-th round state variable of `Twinkle-P`. Specifically, we set the four input bits $S^0[0][0][0]$, $S^0[1][0][0]$, $S^0[2][0][0]$, and $S^0[3][0][0]$ as active. Thus, by activating four input bits, we derive a 4-round integral distinguisher that exhibits 1152 balanced output bits (the detailed result can be found in Table 6). Additionally, we identify a 5-round integral distinguisher by setting 12 input bits as active bits. This distinguisher results in 38 balanced bits at the output, leading to a data complexity of 2^{12}. A summary of the 5-round integral distinguisher is provided in Table 7.

We also applied the division property to 6 rounds. The tool did not yield any integral distinguishers when fewer active bits (e.g., 4/12/36 bits) were set in the input; in this scenario, all output bits are unknown after 6 rounds. However, increasing the number of active bits in the input to search for 6-round integral distinguishers based on the division property significantly increased the running time. As a result, we could not find any 6-round integral distinguisher based on the division property.

We also employed the division property-based method proposed in [12] to compute an upper bound on the algebraic degree. Using this approach, we determine that the algebraic degree of `Twinkle-P` is upper bounded by 27, 81, and 243 for 3, 4, and 5 rounds, respectively, which aligns with the trivial bounds.

5.6 Differential-Linear Distinguishers

We observed that the designers of `Twinkle` did not analyze `Twinkle-P` against differential-linear (DL) distinguishers, even though DL distinguishers are often among the most effective techniques against cryptographic permutations. Here, we fill this gap by providing strong DL distinguishers for up to 6 rounds of Twinkle-P.

For $X = (x_0, x_1, \ldots, x_{n-1})$ and $Y = (y_0, y_1, \ldots, y_{n-1})$ in $\mathbb{F}_2^n$, we define the dot product as $X \cdot Y = \sum_{i=0}^{n-1} x_i \cdot y_i$. Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a vectorial Boolean function. For a given input difference Δ and output linear mask Γ, the correlation is defined as

$$\mathrm{corr}(\Delta \to \Gamma) = 2^{-n} \cdot \sum_{X \in \mathbb{F}_2^n} (-1)^{\Gamma \cdot (F(X) + F(X+\Delta))}, \tag{1}$$

where "+" denotes addition in $\mathbb{F}_2^n$ (i.e., bitwise XOR). Given a sample space $\mathcal{S} \subseteq \mathbb{F}_2^n$, the empirical correlation over $\mathcal{S}$ is defined by

$$\mathrm{corr}_{\mathcal{S}}(\Delta \to \Gamma) = 2^{-|\mathcal{S}|} \cdot \sum_{X \in \mathcal{S}} (-1)^{\Gamma \cdot (F(X) + F(X+\Delta))}. \tag{2}$$

The data complexity of a DL distinguisher with correlation c is in $\mathcal{O}(c^{-2})$.

To search for differential-linear (DL) distinguishers for Twinkle-P, we use a state-of-the-art technique proposed by Hadipour et al. in CRYPTO 2024 [20]. According to this method, we split the permutation E into three parts: $E = E_\ell \circ E_m \circ E_d$. We model the forward probabilistic differential propagation through E_d and the backward probabilistic linear propagation through E_ℓ. These sub-models allow us to find regular differential and linear trails for E_d and E_ℓ, respectively. We also model the deterministic difference propagation (forward) and deterministic linear mask propagation (backward) through E_m to encode the amount of overlap, measured by the number of shared active bits, between the differential and linear propagations in the middle part. Finally, we combine all these sub-models into a unified CP model, connect the junctions, and set the objective function to minimize the sum of the scaled weight of the differential trail through E_d, the scaled overlap weight through E_m, and the scaled weight of the linear trail through E_ℓ. For more details on this method, we refer to [20].

Figure 11a, Figure 11b, and Figure 11c illustrate the DL distinguishers for 4, 5, and 6 rounds of Twinkle-P, respectively. The colors and symbols in these figures follow the same convention as those used for the ID and ZC distinguishers. Let X_i denote the internal state before the S-box in round i, and let ΔX_i and ΓX_i represent the corresponding difference and linear mask, respectively. The input differences and output masks of the DL distinguishers are summarized in Table 8, Figure 8, and Figure 9.

For 4 rounds of Twinkle-P, we discovered a deterministic DL distinguisher, as shown in Figure 11a. Table 8 represents the input difference and output mask of this distinguisher. As shown in Figure 11a, the overlap between the deterministic forward differential and deterministic backward linear propagations for the output of S-box layer is zero. As a result, due to the (bit-wise) switching effect [20], the correlation of this 4-round DL distinguisher should be one and it is a deterministic distinguisher. We verified the correctness of this distinguisher in practice to ensure the soundness of both our model and the result.

For 5 rounds, we decomposed the permutation into a $1 + 3 + 1$ round structure. Figure 11b illustrates this distinguisher, and Figure 8 briefly describes its specification. The first round is covered by a pure differential distinguisher (E_d), the middle 3 rounds are handled by a combined DL distinguisher (E_m), and the last round is covered by a pure linear distinguisher (E_ℓ). The probability of the differential distinguisher over E_d is $p = \Pr(\varDelta X_0 \to \varDelta X_1) = 2^{-1.415}$. As shown in Figure 11b, the overlap between the deterministic differential and linear propagations through E_m is zero, which implies that the correlation over the middle part is $r = \mathrm{corr}(\varDelta X_1 \to \varGamma X_4) = 1$. Moreover, the squared correlation of the linear distinguisher over E_ℓ is $q^2 = \mathrm{corr}^2(\varGamma X_4 \to \varGamma X_5) = 2^{-2}$. As a result, we estimate the total correlation of the 5-round DL distinguisher as $c = prq^2 = 2^{-3.415}$. We conducted experiments and observed that the actual correlation is even higher in practice, with $c = 2^{-2.85}$ on average. Consequently, the data complexity of this distinguisher is approximately $2^{5.70}$.

For 6 rounds, we decomposed the permutation into a 1+4+1 round structure, as shown in Figure 11c. We also specified the input difference and output mask of this distinguisher in Figure 9. We have $p = \Pr(\varDelta X_0 \to \varDelta X_1) = 2^{-18.660}$ and $q^2 = \mathrm{corr}^2(\varGamma X_5 \to \varGamma X_6) = 2^{-18}$. In addition, as seen in Figure 11c, the overlap between the deterministic differential and linear propagations through the middle 4 rounds (E_m) is zero, which implies that the correlation of the small DL distinguisher in the middle is $r = \mathrm{corr}(\varDelta X_1 \to \varGamma X_5) = 1$. As a result, we estimate the total correlation as $c = prq^2 = 2^{-36.66}$. Hence, the data complexity of our 6-round DL distinguisher is $2^{73.32}$. Moreover, since `Twinkle-P` preserves differential, and linear trails under rotation along the z-axis, we can generate 80 distinct DL distinguishers with almost the same correlation from each original one.

6 Conclusion

In this paper, we presented cryptanalysis against `Twinkle`. Our attacks on `Twinkle-AE` targeted variants that aim for a higher confidentiality than integrity. The attacker could make $O(2^t)$ queries to pass the verification, which made our attacks work in the nonce-respecting manner. We showed that c-bit authentication key is recovered only with $O(2^t)$ queries, which further allowed universal forgery attacks, i.e. for any plaintext, the attacker could generate the ciphertext that could be successfully decrypted to the target plaintext. We also pointed out that $O(1)$ attack is possible in the nonce-misuse setting, and key-commitment security can be attacked with a negligible cost. We also presented cryptanalyses on the underlying permutation `Twinkle-P` as a standalone permutation, and applied most well-known cryptanalytic approaches such as impossible differential, zero-correlation, integral, and differential-linear distinguishers by developing automated analytic tools. Our differential-linear distinguishers reach 6 rounds with $2^{73.32}$ complexity, which surpasses the round bounds evaluated by the designers. We stress that none of the results really change the current state of Twinkle because the attacks on the mode are only applicable in a degenerate case that the designers might want to disallow and the distinguishers on the permutation are unlikely to have any significant impact on the ciphers. Moreover, we note that `Twinkle-AE`-b and `Twinkle`-PA remain secure, and the versions we attacked would also be secure if the claimed confidentiality level matched the integrity level.

Acknowledgments. This work was initiated during ASK 2024. We want to appreciate the organizers of ASK 2024 for their efforts in creating a helpful and inspiring environment.

References

1. Caesar: Competition for authenticated encryption: Security, applicability, and robustness. `https://competitions.cr.yp.to/caesar.html` (2014)
2. Amd secure encrypted virtualization (sev) — amd (2019), `https://www.amd.com/en/developer/sev.html`
3. Nist lightweight cryptography competition. `https://csrc.nist.gov/projects/lightweight-cryptography/finalists` (2021)
4. Anand, R., Banik, S., Caforio, A., Ishikawa, T., Isobe, T., Liu, F., Minematsu, K., Rahman, M., Sakamoto, K.: Gleeok: A family of low-latency prfs and its applications to authenticated encryption. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(2), 545–587 (2024), `10.46586/tches.v2024.i2.545-587`
5. Architecture & Technology Group: Arm CCA Security Model 1.0. `https://developer.arm.com/documentation/DEN0096/latest/` (August 2021), document number: DEN0096
6. ARM Holdings: Introduction to pac (2021), `https://developer.arm.com/documentation/109576/0100/Pointer-Authentication-Code/Introduction-to-PAC`
7. Avanzi, R.: 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). `https://doi.org/10.13154/tosc.v2017.i1.4-44`
8. Banik, S., Isobe, T., Liu, F., Minematsu, K., Sakamoto, K.: Orthros: A low-latency PRF. IACR Trans. Symmetric Cryptol. **2021**(1), 37–77 (2021), `10.46586/tosc.v2021.i1.37-77`
9. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer (2016). `https://doi.org/10.1007/978-3-662-53008-5_5`
10. Belkheyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: Bipbip: A low-latency tweakable block cipher with small dimensions. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 326–368 (2023), `10.46586/tches.v2023.i1.326-368`
11. Bellare, M., Hoang, V.T.: Efficient schemes for committing authenticated encryption. In: EUROCRYPT 2022. LNCS, vol. 13276, pp. 845–875 (2022)
12. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli :

A cross-platform permutation. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 299–320. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_15

13. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications (full version). IACR Cryptol. ePrint Arch. p. 529 (2012), http://eprint.iacr.org/2012/529
14. Canale, F., Güneysu, T., Leander, G., Thoma, J.P., Todo, Y., Ueno, R.: SCARF - A low-latency block cipher for secure cache-randomization. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 1937–1954. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/canale
15. Cui, T., Chen, S., Jia, K., Fu, K., Wang, M.: New automatic search tool for impossible differentials and zero-correlation linear approximations. IACR Cryptology ePrint Archive, Report 2016/689 (2016), https://eprint.iacr.org/2016/689
16. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: CRYPTO 2018. LNCS, vol. 10991, pp. 155–186 (2018)
17. Elbaz, R., Champagne, D., Gebotys, C.H., Lee, R.B., Potlapally, N.R., Torres, L.: Hardware mechanisms for memory authentication: A survey of existing techniques and engines. Trans. Comput. Sci. **4**, 1–22 (2009). https://doi.org/10.1007/978-3-642-01004-0_1
18. Farshim, P., Orlandi, C., Rosie, R.: Security of symmetric primitives under incorrect usage of keys. IACR Trans. Symmetric Cryptol. **2017**(1), 449–473 (2017)
19. Gueron, S.: Memory encryption for general-purpose processors. IEEE Secur. Priv. **14**(6), 54–62 (2016), https://doi.org/10.1109/MSP.2016.124
20. Hadipour, H., Derbez, P., and, M.E.: Revisiting differential-linear attacks via a boomerang perspective with application to AES, Ascon, CLEFIA, SKINNY, PRESENT, KNOT, TWINE, WARP, LBlock, Simeck, and SERPENT pp. 38–72 (2024). https://doi.org/10.1007/978-3-031-68385-5_2
21. Hadipour, H., Gerhalter, S., Sadeghi, S., Eichlseder, M.: Improved search for integral, impossible differential and zero-correlation attacks application to ascon, forkskinny, skinny, mantis, PRESENT and qarmav2. IACR Trans. Symmetric Cryptol. **2024**(1), 234–325 (2024). https://doi.org/10.46586/TOSC.V2024.I1.234-325
22. Hadipour, H., Nageler, M., Eichlseder, M.: Throwing boomerangs into feistel structures application to clefia, warp, lblock, lblock-s and TWINE. IACR Trans. Symmetric Cryptol. **2022**(3), 271–302 (2022). https://doi.org/10.46586/tosc.v2022.i3.271-302
23. Hadipour, H., Sadeghi, S., Eichlseder, M.: Finding the impossible: Automated search for full impossible differential, zero-correlation, and integral attacks. In: EUROCRYPT 2023. LNCS, vol. 14007, pp. 128–157. Springer (2023). https://doi.org/10.1007/978-3-031-30634-1_5
24. Hosoyamada, A., Inoue, A., Ito, R., Iwata, T., Minematsu, K., Sibleyras, F., Todo, Y.: Cryptanalysis of rocca and feasibility of its security claim. IACR Trans. Symmetric Cryptol. **2022**(3), 123–151 (2022). https://doi.org/10.46586/TOSC.V2022.I3.123-151
25. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 510–545 (2021). https://doi.org/10.46586/TCHES.V2021.I4.510-545

26. LeMay, M., Rakshit, J., Deutsch, S., Durham, D.M., Ghosh, S., Nori, A., Gaur, J., Weiler, A., Sultana, S., Grewal, K., Subramoney, S.: Cryptographic capability computing. In: MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021. pp. 253–267. ACM (2021). https://doi.org/10.1145/3466752.3480076
27. Len, J., Grubbs, P., Ristenpart, T.: Partitioning oracle attacks. In: USENIX Security 2021. pp. 195–212 (2021)
28. Sakamoto, K., Liu, F., Nakano, Y., Kiyomoto, S., Isobe, T.: Rocca: An efficient aes-based encryption scheme for beyond 5g. IACR Trans. Symmetric Cryptol. **2021**(2), 1–30 (2021). https://doi.org/10.46586/TOSC.V2021.I2.1-30, full version is available on IACR Cryptol. ePrint Arch. 2022/116.
29. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects. In: EUROCRYPT 2017. LNCS, vol. 10212, pp. 185–215. Springer (2017). https://doi.org/10.1007/978-3-319-56617-7_7
30. Sun, B., Liu, Z., Rijmen, V., Li, R., Cheng, L., Wang, Q., AlKhzaimi, H., Li, C.: Links among impossible differential, integral and zero correlation linear cryptanalysis. In: CRYPTO 2015. LNCS, vol. 9215, pp. 95–115. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_5
31. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_12, https://doi.org/10.1007/978-3-662-46800-5_12
32. Todo, Y., Morii, M.: Bit-based division property and application to simon family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_18
33. Wang, J., Huang, T., Wu, S., Liu, Z.: Twinkle: A family of low-latency schemes for authenticated encryption and pointer authentication. IACR Communications in Cryptology **1**(2) (2024). https://doi.org/10.62056/a3n59qgxq
34. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. **22**(3), 265–279 (1981). https://doi.org/10.1016/0022-0000(81)90033-7
35. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678 (2016). https://doi.org/10.1007/978-3-662-53887-6_24

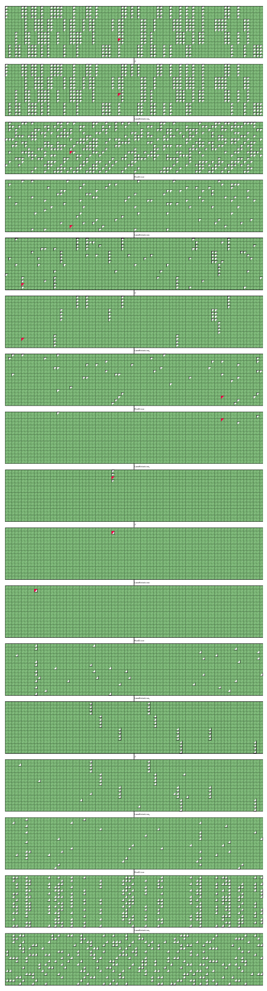
7 Modeling the S-box Using the S-box Analyzer

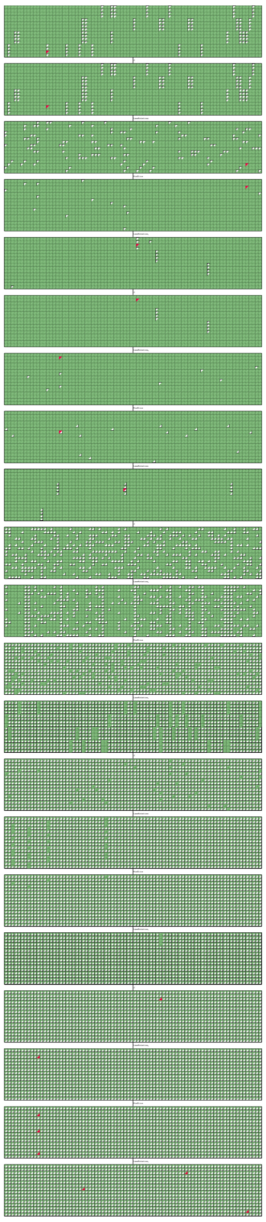
```
# Import the S-box Analyzer and define the S-box
from sboxanalyzer import *
sb = SboxAnalyzer([0x0,0x3,0x5,0xd,0x6,0xf,0xa,0x8,0xb,0x4,0xe,0x2,0x9,0xc,0x7,0x1])
# Model differential propagation with probabilities
cnf, milp, cp = sb.minimized_diff_constraints()
Number of constraints: 58
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0
Weight: 3.0000 p0 + 2.0000 p1 + 1.4150 p2

# Model forward differential propagation without probabilities
cnf, milp,cp = sb.minimized_diff_constraints(subtable='star')
Number of constraints: 39
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0

# Model backward differential propagation without probabilities
cnf, milp, cp = sbi.minimized_diff_constraints(subtable='star')
Number of constraints: 40
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0

# Model deterministic differential propagation (forward)
temp = sb.encode_deterministic_differential_behavior()
cp = sb.generate_cp_constraints(temp); print(cp)
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0
if(a0==0/\a1==0/\a2==0/\a3==0)then(b0=0/\b1=0/\b2=0/\b3=0)
elseif(a0==0/\a1==0/\a2==1/\a3==0)then(b0=-1/\b1=1/\b2=-1/\b3=-1)
elseif(a0==0/\a1==1/\a2==0/\a3==1)then(b0=-1/\b1=1/\b2=-1/\b3=1)
elseif(a0==0/\a1==1/\a2==1/\a3==1)then(b0=-1/\b1=0/\b2=-1/\b3=-1)
elseif(a0==1/\a1==0/\a2==0/\a3==0)then(b0=-1/\b1=-1/\b2=-1/\b3=1)
elseif(a0==1/\a1==1/\a2==0/\a3==1)then(b0=-1/\b1=-1/\b2=-1/\b3=0)
else(b0=-1/\b1=-1/\b2=-1/\b3=-1)
endif

# Model deterministic differential propagation (backward)
temp = sbi.encode_deterministic_differential_behavior()
cp = sbi.generate_cp_constraints(temp); print(cp)
Input:a0||a1||a2||a3;msb:a0
Output:b0||b1||b2||b3;msb:b0
if(a0==0/\a1==0/\a2==0/\a3==0)then(b0=0/\b1=0/\b2=0/\b3=0)
elseif(a0==0/\a1==0/\a2==0/\a3==1)then(b0=1/\b1=-1/\b2=1/\b3=-1)
elseif(a0==0/\a1==1/\a2==0/\a3==0)then(b0=1/\b1=-1/\b2=-1/\b3=-1)
elseif(a0==0/\a1==1/\a2==0/\a3==1)then(b0=0/\b1=-1/\b2=-1/\b3=-1)
elseif(a0==1/\a1==1/\a2==1/\a3==0)then(b0=-1/\b1=-1/\b2=1/\b3=-1)
elseif(a0==1/\a1==1/\a2==1/\a3==1)then(b0=-1/\b1=-1/\b2=0/\b3=-1)
else(b0=-1/\b1=-1/\b2=-1/\b3=-1)
endif
```

Listing 1.1: Encoding differential behavior of S-box

```
# Import the S-box Analyzer and define the S-box
from sboxanalyzer import *
sb = SboxAnalyzer([0x0,0x3,0x5,0xd,0x6,0xf,0xa,0x8,0xb,0x4,0xe,0x2,0x9,0xc,0x7,0x1])
sbi = SboxAnalyzer(sb.inverse())
# Model forward linear mask propagation with correlations
cnf, milp, cp = sb.minimized_linear_constraints()
Number of constraints: 39
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0
Weight: 4.0000 p0 + 2.0000 p1

# Model forward linear mask propagation without correlations
cnf, milp, cp = sb.minimized_linear_constraints(subtable='star')
Number of constraints: 32
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0

# Model backward linear mask propagation withoout correlations
cnf, milp,cp = sbi.minimized_linear_constraints(subtable='star')
Number of constraints: 33
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0

# Model deterministic linear mask propagation (forward)
temp = sb.encode_deterministic_linear_behavior()
cp = sb.generate_cp_constraints(temp); print(cp)
Input:a0||a1||a2||a3;msb:a0
Output:b0||b1||b2||b3;msb:b0
if(a0==0/\a1==0/\a2==0/\a3==0)then(b0=0/\b1=0/\b2=0/\b3=0)
elseif(a0==0/\a1==0/\a2==1/\a3==0)then(b0=-1/\b1=-1/\b2=-1/\b3=1)
elseif(a0==1/\a1==0/\a2==0/\a3==0)then(b0=-1/\b1=1/\b2=-1/\b3=1)
elseif(a0==1/\a1==0/\a2==1/\a3==0)then(b0=-1/\b1=-1/\b2=-1/\b3=0)
else(b0=-1/\b1=-1/\b2=-1/\b3=-1)
endif

# Model deterministic linear mask propagation (backward)
temp = sbi.encode_deterministic_linear_behavior()
cp = sbi.generate_cp_constraints(temp); print(cp)
Input:a0||a1||a2||a3;msb:a0
Output:b0||b1||b2||b3;msb:b0
if(a0==0/\a1==0/\a2==0/\a3==0)then(b0=0/\b1=0/\b2=0/\b3=0)
elseif(a0==0/\a1==0/\a2==0/\a3==1)then(b0=1/\b1=-1/\b2=-1/\b3=-1)
elseif(a0==0/\a1==1/\a2==0/\a3==0)then(b0=-1/\b1=-1/\b2=1/\b3=-1)
else(b0=-1/\b1=-1/\b2=-1/\b3=-1)
endif

# Model monomial propagation in forward direction (MPT)
cnf, milp, cp = sb.minimized_integral_constraints()
Number of constraints: 38
Input: a0||a1||a2||a3; msb: a0
Output: b0||b1||b2||b3; msb: b0
```

Listing 1.2: Encoding linear and integral behaviors of S-box


(a) ID distinguisher for 4 rounds of `Twinkle-P`


(b) ID distinguisher for 5 rounds of `Twinkle-P`

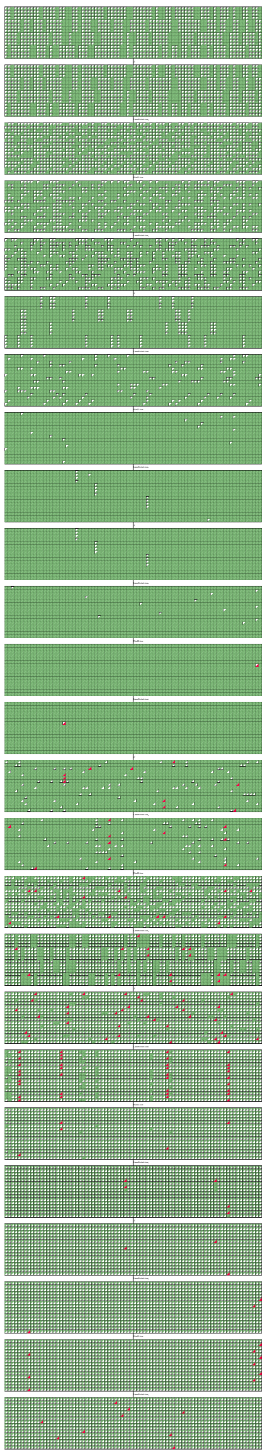

(c) ID distinguisher for 6 rounds of `Twinkle-P`

Fig. 5: ID distinguishers

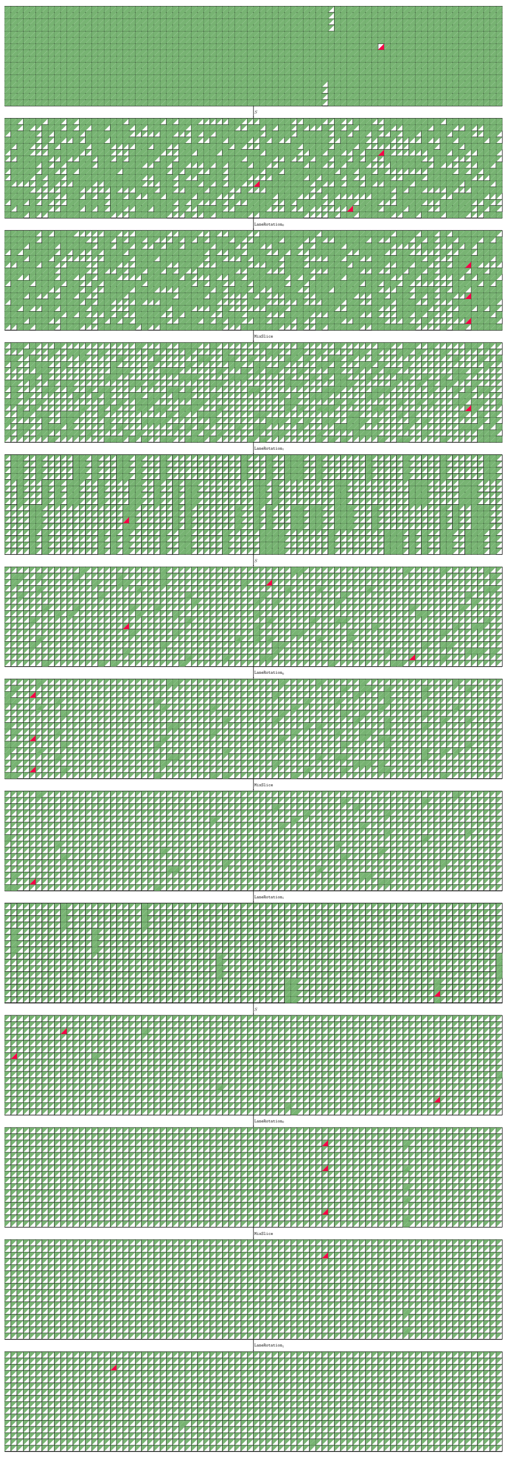
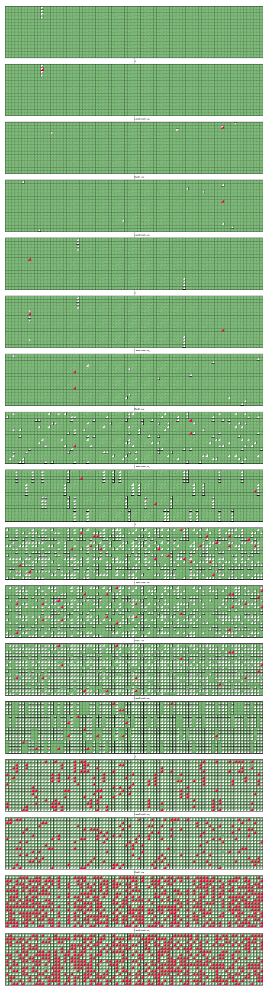
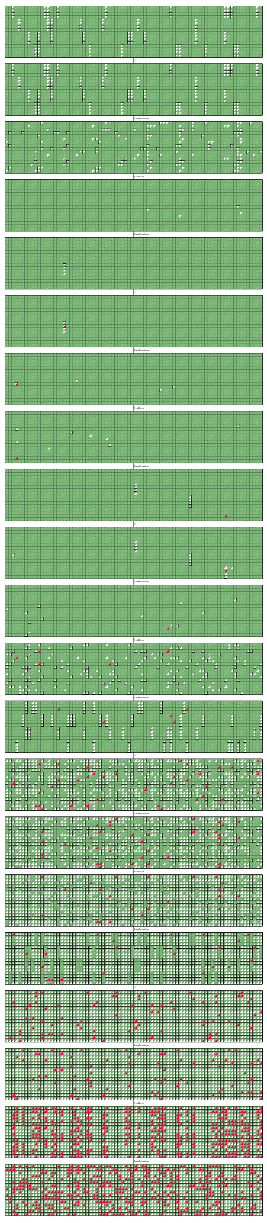


Fig. 6: ZC-based Integral distinguisher for 3 rounds of `Twinkle-P`


(a) ZC distinguisher for 4 rounds of `Twinkle-P`


(b) ZC distinguisher for 5 rounds of `Twinkle-P`

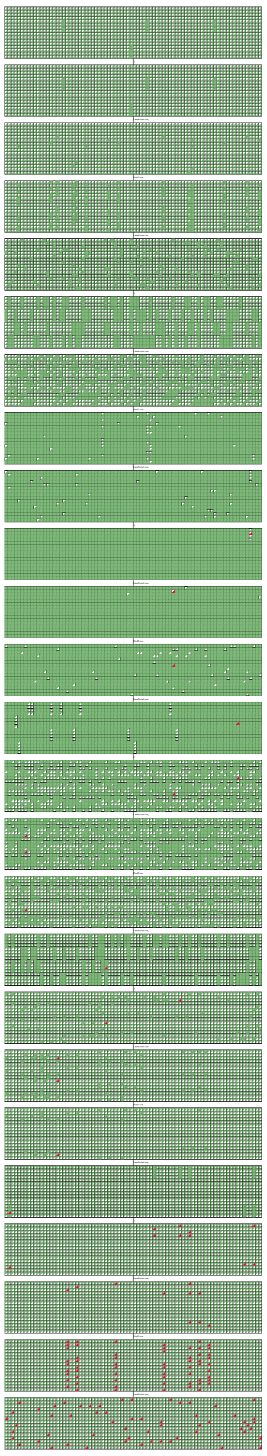

(c) ZC distinguisher for 6 rounds of `Twinkle-P`

Fig. 7: ZC distinguishers and ZC-based integral distinguishers

Input Division Property
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
Output Division Property
bbbbbbbbbbbbbbbbbb?bb?bbbbbbbbbb?bb?bbbbbbb?bbbb?bbbbbbbbbbbbb?bb?bbb?bbbbbb?bb?
bbbbbb?bbbbbbbbbbb?bbbb?bbbb?bbbbbb?b?bbbbb?bbbbbbbbbbbbbbbbbbbbbbbbbb?bbbbbbbbb
?bbbbbbbbbbbbb?bbbbbbbbbb?bbbbbb?bbbbbbbbb?bbbbbbbbbbbbbbb?bb?bbbbbbbbbbbbbbb?bb
bbbbbbbbbbbbbbbbbbb?bbbbbb?bbbbbbbbbbbbbbb?bbbbbbbbbbb?bbbbbbbbbbbbbbb??bbbbbbb?
bbbbb?bbbbbbbbbbbbbbbbbbbbbbbbbbbbb?bb?bbb?bbbbb?bbbb?bbbbbbbb?bbbbbb?b?bbbbbbbb
bbbbbb?bbbbbbbbbbb?bb?bbbbbbbbbb?bbbbbbbbbb?bbbbbbbbbbbbbbbbbb?bb?bbb?bbbbbb?bb?
bbbbbbbbbbbbbbbbbb?b?bbbbbb?bbb?bbbbbbbbb?bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb?bbbbb
?bbbbbbbbbbbbb?bbbbbbbbbb?bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb?bb?bbbbbbbbbbbbbbb?bb
bbbbb?bbbbbbbbbbbb??bbb?bbbbbbbbbbb?b??bbb??bbbbbbbbbbbbbbbbbbbbbbbbb???bbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb?bbbbbbbbbbbb?bbbbbbbbbbbbb?bbbbbbbbbbbbbbbbb
bbbbbb?bbbbbbbbbbb?b?bbbb?bbbbbbbbbbbbbbbb?bbbbbbbbbbb?bbbbbbbbbbbbbbb??bb?bbbb?
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb?bbbbbbbbb?bbbbbbbbbb?bbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbb?bb?b?bbbbbbbb?bb?b?bbbbb?bbbbbbbbbbbbbbbbbbb?bb?bbb??bbbbb?bb?
bbbbb?bbbbbbbbbbbbbbbbbbbbbb?bbbbbbbbb?bbb?bbbbbbbbbbbbbbbbbbbbbbbbbb?b?bbbbbbbb
?bbbbbbbbbbbbb?bbbbbbbbbb??bbbbbbbbbbbbbbbb?bbbbbbbbbb?bbb?bb?bbbbbbbb??bbbbb?b?
bbbbbbbbbbbbbbbbbb?b?bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb?bbbbbbbbbbbbbbbbbbb?bbbbb

Table 6: Division-property-based integral distinguisher for 4 rounds. The symbols “1” and “0” at the input indicate the positions of active and constant bits, respectively. At the output, “b” and “?” denote balanced and unknown bits, respectively.

Input Division Property
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
Output Division Property
??
??b???b?????????????????????
??
??????????b?????????????????????????b????????????????????b?????b????????????????
??
????????????????????????b??b??????????????????b????????????????????b????????????
??
??????b??????????????????????????????????????b???b????b?????????????????????????
??b?????????????????????????
?????????????????????????b???b???????b
??????????b?????????????????????????b???
??????????????????b?????????????????????b??????????????????????????b??????b????b
??
????b?b??b??b???????????????b?????????
??
????b?????b?????????b???????????????b??b??????????????????????b?b????b??????????

Table 7: Division-property-based integral distinguisher for 5 rounds. The symbols “1” and “0” at the input indicate the positions of active and constant bits, respectively. At the output, “b” and “?” denote balanced and unknown bits, respectively.

ΔX_0
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000010000000000000000000000000000000
00000000000000000000000000000000000000000000000010000000000000000000000000000000
00000000000000000000000000000000000000000000000010000000000000000000000000000000
ΓX_4
00000100000001001000000000000000000000000000010000000011000000010000000000000000
00000000000000000100000100000000000010000000000000001000000000000000000010000000000
00100000001000000000000000000010000000000000000000000000000000000100000000000000100000
00000100000000000000000100000000000000001000000000000010100000000000000000000000100
00000000000000000000000000000000000000000000100000000000000001000000000000001000
01000000000000000000000001000000000100100000000000000000000000000000000000110000000
00000000100000000000000000000000000100000000000000100000010000000000000000000000000
00000000001001000000000000000101000000000000100000001000000010000000000000000001000
00000000100001000000001001000000000000000000000001000000000000100000000001000000000000000000
00000000011000000010000000000000000000000000000000100000000000100000000000000100000000000000
00000001000000000000010000000000000000001000000000000000000001000000010000000000
10000000100000000000000000100000001000000000000000000000000000000101000000000000000000001
00000000000000010000000000000000000000000000000100000000000000000000000000000000
01010000000000000000001000000010000000000000000000000000001000001000000000001000000000000
00000000000000000000000000000000000000000000000000000000000001001000000000000010
11000000010000000000000000010000010000000100000000000000010100000000000001000000

Table 8: Deterministic DL distinguisher for 4 rounds

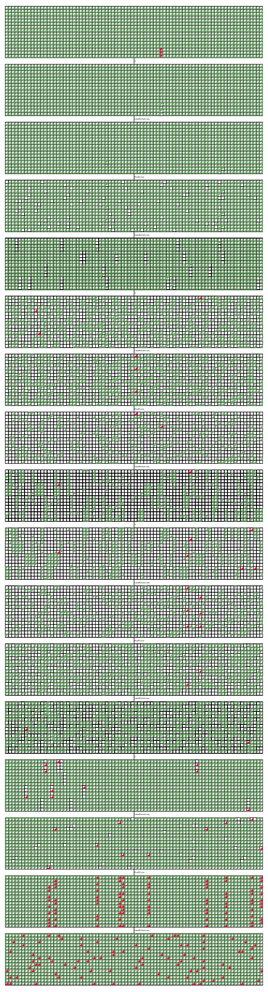
ΔX_0
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000001000000000000000000000000000000000000000
00000000000000000000000000000000000000001000000000000000000000000000000000000000
00000000000000000000000000000000000000001000000000000000000000000000000000000000
00000000000000000000000000000000000000001000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
ΔX_1
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000010000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000100000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000001
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
ΓX_4
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000100000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
ΓX_5
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000001000000000000000000000000010000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000001000000000000000000000000100000000000000000000000000
00000000000000000000000000000000000100000000000000000000000000000000000000000000
00000000000000000000000010000000000000000000000000100000000000000000000000000000
00000000000000000000000000000000010000000000000000000000000000000000000000000000
00000000000000000000000000010000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000100000000000000000000000000000000000000
00000000000010000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000010000000000000000000000
00000010000000000000000000000000001000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000001000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000010
00000000000000000000000000000000100000000000000000000000001000000000000000000000

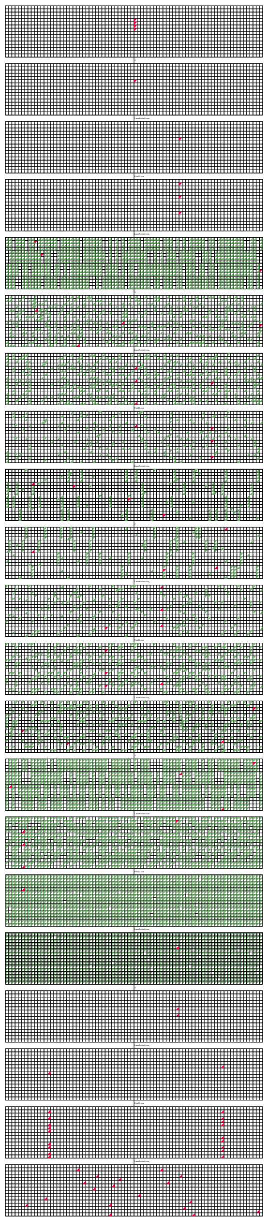
Fig. 8: DL distinguisher for 5 rounds

ΔX_0
00001000000000000000000000000001000000000000000000000000000000000000000000000000
00000000000000000000000000000001000000000000000000000000000000000000000000000000
00001000000000000000000000000001000000000000000000000000000000000000000000000000
00000000000000000000000000000001000000000000010000000000000000000000000000000000
00000000000000000000000000000000000000000000000100000000000000000000000001000000
00000000000000000000000000000000000000000000000000000000000000000000000001000000
00000000000000000000000000000000000000000000000100000000000000000000000001000000
00000000000000000000000000000000000000000000000000000000000000000000000001000000
00000000000000000000001000000000000000000000000000000000000000000000000000000000
00000000000000000000001000000000000000000000000000000100000000000000000000000000
00000000000000000000001000000000000000000000000000000000000000000000000000000000
00000000000000000000001000000000000000000000000000000000000000000000000000000000
00000000000000000000000001000000000000000000000000000000000000000000000000000000
00000000000000000000000011000000000000000000000000000000000000000000000000000000
00000000000000000000000001000000000000000000000000000000000000000000000000000000
00000000000000000000000001000000000000000000000000000000000000000000000000000000
ΔX_1
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000010000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
ΓX_5
00000000000000000000000000000000000000000000000000010000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00100000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000100000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000010000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000010000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000100000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000100000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000001000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000001
ΓX_6
00000000000000001000000000110000000000000000100000000000000000100110000000000000000001
10000000010000001000000000000000000000100000010000000000010000000000000000100000000001000000101
01000001000000000000100000010000000000000000000001010000000001100000000000000000000000000
00010110000000000100000000000000000000000000000001000000010000000100000000001000000110000000100000000000000000000000000000000010
00010100010000000000000000000000000000010000000001000000000100000101000100000000000000001000000000000000000000000
01100000000000000000110000000000000000000000000000100010001100000100000000000000000000010000000000000000000000000000010000011
00100000000100000000000000000000000000000000000000100000000000010000000000011000000000000000000000000000
00000100000000000100000000110000000000000000000000000000000000000000101000010110000000000000000010000000000000000000000000000100000
10000000100000000000000000100000000000000000000000000000001000000000000000000000000000000000001100000000
01000000101000000001100000000000000000000000000000001100000000000100000000000100010100010001100000000000000000001000000000
00000000000000000000000000000000000000000000000000000000000000100011000001000000000000000000000000000000000000000000010000000000000000001
01000110000000000000000100000000000000000001100000000000100000000000000000010100000000000110000000000000000000000000000000000001000000001
10000000000000000000000000000000000000000000000000000000001010001000000000000000100000000000000000000000000000010000000000000000001000000000000000000000000000000000000000000
00001000000000000000010000000000000000000000000101001001100000000000000000000000000000000000000001000000000000000000000000000001100000000000000000000000000000000000000001000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000001000100000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000110110000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010000000000000000000000000
00100001000000000000100000001000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100011000000000000000000000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000000000

Fig. 9: DL distinguisher for 6 rounds

Fig. 10: Specification of DL distinguishers for `Twinkle-P`


(a) DL distinguisher for 4 rounds of `Twinkle-P`


(b) DL distinguisher for 5 rounds of `Twinkle-P`

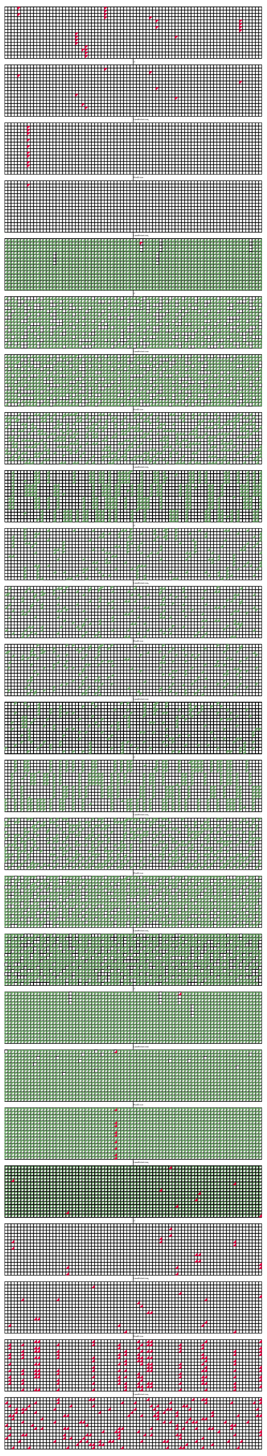

(c) DL distinguisher for 6 rounds of `Twinkle-P`

Fig. 11: DL distinguishers