ChinaRxiv

AI translation · View original & related papers at
chinarxiv.org/items/chinaxiv-201805.00202

Meet-in-the-Middle Attack on 10-Round Midori128 (Postprint)

Authors: Liu Ya, Diao Qianqian, Li Wei, Liu Zhiqiang

Date: 2018-05-20T00:00:00+00:00

Abstract

Lightweight block ciphers are widely deployed in resource-constrained smart devices for data security protection due to their low implementation cost in both hardware and software as well as low power consumption. Midori is a lightweight block cipher algorithm presented at the 2015 ASIACRYPT conference, featuring two block sizes of 64-bit and 128-bit, designated as Midori64 and Midori128, respectively. To date, no results have been reported on Midori128' s resistance against meet-in-the-middle attacks. By investigating the fundamental structure of the Midori128 algorithm and the characteristics of its key schedule, and incorporating differential enumeration and related-key filtering techniques, we construct a 7-round meet-in-the-middle distinguisher. Subsequently, by prepending one round and appending two rounds to this distinguisher, and employing a time-memory trade-off approach, we present the first meet-in-the-middle attack on the 10-round Midori128 algorithm. The complete attack necessitates a time complexity of 2126.5 10-round Midori128 encryptions, a data complexity of 2125 chosen plaintexts, and a memory complexity of 2105 128-bit blocks, representing the inaugural meet-in-the-middle attack on Midori128.

Full Text

Meet-in-the-Middle Attacks on 10-Round Midori128

Liu Ya[1] ,[1] ,[2]†, **Diao Qianqian**[1] ,[3], **Li Wei** , , **Liu Zhiqiang**[2]
[1](a. School of Optical-Electrical & Computer Engineering, b. Shanghai Key Lab of Modern Optical System, University of Shanghai for Science & Technology, Shanghai 200093, China; [2]. Dept. of Computer Science & Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; [3]. Shanghai Viewsource Information Science & Technology Co., Ltd, Shanghai 200240, China; . School of Computer Science & Technology, Donghua University, Shanghai 201620, China; . Shanghai Key Laboratory of Integrate Administration Security, Shanghai 200240, China)

Abstract: Lightweight block ciphers are widely deployed in resource-constrained smart devices to protect data security due to their low software and hardware implementation costs and power consumption. Midori is a lightweight block cipher presented at ASIACRYPT 2015, with two block size variants: 64-bit (Midori64) and 128-bit (Midori128). To date, no results have been published regarding meet-in-the-middle attacks on Midori128. By studying the fundamental structure and key schedule characteristics of Midori128, and employing differential enumeration combined with key-dependent sieve techniques, we construct a 7-round meet-in-the-middle distinguisher. We then extend this distinguisher by adding one round at the beginning and two rounds at the end, proposing the first meet-in-the-middle attack on 10-round Midori128 using a time-memory tradeoff approach. The attack requires a time complexity of $2^{126.5}$ 10-round encryptions, a data complexity of 2^{125} chosen plaintexts, and a memory complexity of 2^{105} 128-bit blocks, representing the first meet-in-the-middle analysis of Midori128.

Keywords: block cipher; meet-in-the-middle attack; Midori128

0 Introduction

With the development of Internet technology and the construction of smart cities, resource-constrained devices such as RFID tags and sensors are increasingly deployed in IoT, cloud computing, and smart city applications. Ensuring data security on these devices has become a critical research challenge. Lightweight block ciphers, characterized by fast software and hardware implementation efficiency and low power consumption, are widely adopted for data protection in such environments. However, high implementation efficiency inevitably sacrifices some security margins, making it essential to evaluate lightweight block ciphers against various effective cryptanalytic techniques.

In 2015, Banik et al. [1] introduced a new lightweight block cipher called Midori at ASIACRYPT. Based on the SPN structure with 20 rounds and a 128-bit key, Midori supports two block sizes: 64 bits and 128 bits, denoted as Midori64 and Midori128 respectively. The algorithm operates on a 4×4 byte matrix where each cell represents an 8-bit unit.

Owing to its low power consumption and high implementation efficiency, Midori has broad application prospects in practice, and studying its security provides a theoretical foundation for real-world deployment. The security of Midori64 has been evaluated against various attacks including differential cryptanalysis, impossible differential analysis, related-key impossible differential analysis, higher-order differential analysis, meet-in-the-middle attacks, and invariant subspace attacks [2–6]. In contrast, Midori128 has only been assessed through differential analysis, impossible differential analysis, and related-key differential analysis [7–10]. Specifically, references [7,8] applied impossible differential analysis to attack 10-round and 11-round Midori128 without whitening keys; reference [9] used differential analysis against 13-round Midori128; and reference [10] pre-

sented a full-round related-key differential analysis of Midori128. However, no researchers have yet evaluated Midori128' s resistance to meet-in-the-middle attacks, which have proven highly effective in recent years and have yielded the best cryptanalytic results for many prominent ciphers such as AES [11]. Therefore, investigating Midori128' s resistance to meet-in-the-middle attack is both important and timely.

The meet-in-the-middle attack, originally proposed by Diffie and Hellman [12], has been widely applied to hash functions and block ciphers, particularly in AES security analysis [14]. The fundamental principle involves dividing the encryption algorithm E into two parts ($E = E_2 \circ E_1$) or three parts ($E = E_2 \circ E_{mid} \circ E_1$), where E_{mid} contains a multi-round distinguisher. In the three-part variant, a precomputed ordered sequence is first constructed for E_{mid}. Then for a plaintext-ciphertext pair (P, C), the keys for E_1 and E_2 are guessed. If $E_1(P, k_1)$ and $E_2(C, k_2)$ satisfy the precomputed sequence, the key guess is retained; otherwise, it is eliminated. Through multiple plaintext-ciphertext pairs, the correct key can eventually be filtered out.

In 2000, Gilbert et al. [15] utilized a 4-round AES distinguisher to mount a collision attack on 7-round AES. In 2008, Demirci et al. [16] extended the AES meet-in-the-middle distinguisher to 5 rounds and improved the analysis of 8-round AES-256. Subsequently, Dunkelman et al. [17] in 2010 proposed several techniques including differential enumeration to reduce precomputation parameters, enhancing the analysis of 7/8-round AES-192/256. In 2014, Li et al. [18] exploited key schedule properties to improve the 5-round AES distinguisher and presented a meet-in-the-middle attack on 9-round AES-192/256. In 2016, reference [11] employed differential enumeration and key-dependent sieve techniques to construct a 6-round meet-in-the-middle distinguisher, further reducing precomputation parameters and improving the analysis of 10-round AES-256. Additionally, meet-in-the-middle analyses have been applied to LED [19], CLEFIA, Camellia [20], TWINE [21], and LBlock [22].

After careful study of Midori128' s structure and key schedule, we present the first meet-in-the-middle attack on 10-round Midori128. The attack leverages differential enumeration and key-dependent sieve techniques to construct a 7-round distinguisher, then extends it by adding one round at the front and two rounds at the rear.

1.1 Notation

We first establish the notation used throughout this paper:

a) P, C: plaintext and ciphertext

b) rk_i: round key

c) x_i, y_i, z_i, w_i: intermediate state values before SubCell, ShuffleCell, MixColumn, and KeyAdd operations in round i

d) $x_i[j]$: the j-th cell of the state in round i

e) x_i^j: the j-th value in a sequence of x_i

f) $\Delta x_i[j]$: difference defined as $x_i[j] \oplus x'_i[j]$

g) $\overline{x}_i$: inverse of x_i such that $\overline{x}_i = \text{SubCell}^{-1}(x_i)$

h) $\overline{rk}_i$: inverse of round key such that $\overline{rk}_i = \text{MixColumn}^{-1}(rk_i)$

Definition 1 (2-δ-set) [13]: A 2-δ-set is a collection of states where two bytes (active cells) take all possible values while the remaining 14 bytes (fixed cells) remain constant. This set contains $2^{2\times 8}$ elements.

Proposition 1 (Differential Property of S-box): For a given S-box, if the input difference is Δ_i and the output difference is Δ_o, on average there exists one input value x satisfying $S(x) \oplus S(x \oplus \Delta_i) = \Delta_o$.

1.2 Midori Algorithm Description

Midori is a lightweight block cipher based on the SPN structure, presented by Banik et al. at ASIACRYPT 2015. Each round consists of four operations: SubCell, ShuffleCell, MixColumn, and KeyAdd, as shown in [Figure 1: see original paper]. The encryption begins with a pre-whitening key addition, and the final round omits ShuffleCell and MixColumn operations, retaining only SubCell and KeyAdd.

The cipher state is represented as a 4×4 matrix where each cell is an 8-bit byte:

$$S = \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}$$

SubCell: Each byte undergoes a nonlinear S-box substitution.

ShuffleCell: Byte-wise permutation operation: $(r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}) \leftarrow (r_0, r_{10}, r_5, r_{15}, r_{14}, r_4, r_{11}, r_1, r_9, r_3, r_{12}, r_6, r_7, r_{13}, r_2, r_8)$

MixColumn: Each column is multiplied by a 4×4 MDS matrix M:

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

KeyAdd: The round key is XORed with the state.

Key Schedule: The whitening key for the first round and the subkey for the final round are both the 128-bit master key. Intermediate round keys are generated by XORing the master key with round constants: $rk_i = K \oplus \beta_i$ for $1 \leq i \leq 19$, where β_i are constants.

2 Meet-in-the-Middle Attack Introduction

The meet-in-the-middle attack, first proposed by Diffie and Hellman [12], has been extensively applied to hash functions and block ciphers, particularly in AES security analysis [14]. The basic principle has two variants: (1) The encryption algorithm E is split into $E = E_2 \circ E_1$, where E_1 and E_2 use keys k_1 and k_2 respectively. By guessing k_1 and k_2, if for some plaintext-ciphertext pair (P, C) we have $E_1(P, k_1) = E_2(C, k_2)$, the key guess is correct; otherwise, it is wrong. (2) The algorithm is divided into three parts $E = E_2 \circ E_{mid} \circ E_1$, where E_{mid} contains a multi-round distinguisher. A precomputed ordered sequence is first constructed for E_{mid}. Then for a plaintext-ciphertext pair (P, C), the keys (k_1, k_2) for E_1 and E_2 are guessed. If $E_1(P, k_1)$ and $E_2(C, k_2)$ satisfy the precomputed sequence, the key guess is retained; otherwise, it is eliminated.

In 2000, Gilbert et al. [15] used a 4-round AES distinguisher to perform a collision attack on 7-round AES. In 2008, Demirci et al. [16] extended the AES meet-in-the-middle distinguisher to 5 rounds and improved the analysis of 8-round AES-256. Subsequently, Dunkelman et al. [17] in 2010 proposed differential enumeration and other techniques to reduce precomputation parameters, improving the analysis of 7/8-round AES-192/256. In 2014, Li et al. [18] exploited key schedule properties to improve the 5-round AES distinguisher and presented a meet-in-the-middle attack on 9-round AES. In 2016, reference [11] utilized differential enumeration and key-dependent sieve techniques to construct a 6-round meet-in-the-middle distinguisher, further reducing precomputation parameters and improving the analysis of 10-round AES-256. Additionally, references [19-22] presented meet-in-the-middle analyses of LED, CLEFIA, Camellia, TWINE, and LBlock.

3 Meet-in-the-Middle Attack on 10-Round Midori128

We first construct a 7-round meet-in-the-middle distinguisher using differential enumeration and key-dependent sieve techniques, as shown in [Figure 2: see original paper]. We then extend this distinguisher by adding one round at the front and two rounds at the back to mount a meet-in-the-middle attack on 10-round Midori128.

3.1 10-Round Midori128 Distinguisher

In [Figure 2: see original paper], since $r_8[5] = r_8[4] \oplus r_8[6] \oplus r_8[7]$ and $r_8[6] = r_8[4] \oplus r_8[5] \oplus r_8[7]$, we have $r_8[5] \oplus r_8[6] = r_8[6] \oplus r_8[5]$. Let $c_1 = r_8[6] \oplus r_8[5]$ and $c_2 = r_9[5] \oplus r_9[6]$, then $c_2 = c_1 \oplus \Delta r_8[5] \oplus \Delta r_8[6]$.

Proposition 2: Let $r_2[3,12]$ be active bytes. Then $\{r_2^0, r_2^1, \cdots, r_2^{2^{16}-1}\}$ forms a 2-δ-set. Taking $r_2^0[12]$ and 18 distinct values of $r_2^i[3]$ $(i = 1, \cdots, 18)$ such that they satisfy $\Delta r_2^i[3] = \Delta r_2^0[3]$ after the S-box operation, and encrypting them through 7 rounds, if one element in this set satisfies the truncated differential path in [Figure 2: see original paper], then the 144-bit ordered sequence $(\Delta r_{18} \oplus r_{18}^0, \cdots, \Delta r_{18} \oplus r_{18}^0)$ is determined by 46 bytes.

Proof: As shown in [Figure 2: see original paper], when a message pair (r_2^0, r_2^i) satisfies the truncated differential chain, the output difference sequence $(\Delta r_{18} \oplus r_{18}^0, \cdots, \Delta r_{18} \oplus r_{18}^0)$ is determined by the following 46 bytes:

$$r_0[3,12] \parallel r_3[0,2,3,9,10,11] \parallel r_4[0,1,2,3,4,5,7,8,9,10,11,12,13,15] \parallel r_5[0,1,2,3,4,5,6,8,9,10,11,12,13,15]$$

In fact, these 46 bytes can be reduced to 31 bytes:

$$r_0[9] \parallel \Delta r_2[3] \parallel r_3[9,10] \parallel r_4[0,2,3,9,10,11] \parallel \Delta r_5[0,1,2,3,5,7,8,9,10,12,13,14] \parallel r_6[1,3,4,9,11,12] \parallel r_7[4,11$$

Since the differential relations $\Delta r_2[3] = \Delta r_2[12]$ and $\Delta r_2[8,11] = 0$ hold, we can compute $\Delta r_3[9,10]$ and $\Delta r_4[0,2,3,9,10,11]$. Given $r_4[0,2,3,9,10,11]$ and $\Delta r_4[0,2,3,9,10,11]$, we can compute $r_4[0,1,6,8,9,14]$. Similarly, we can compute $\Delta r_5[5]$ from $\Delta r_5[0,1,2,3,5,7,8,9,10,12,13,14]$ and $\Delta r_4[4,11]$. Since $\Delta r_5[12] = \Delta r_2[9]$, we can compute $\Delta r_5[12]$ and $\Delta r_6[1,3,4,9,11,12]$.

Because $\Delta r_5[12] \oplus \Delta r_5[7] = \Delta r_5[13] \oplus \Delta r_5[5] \oplus \Delta r_5[4] \oplus \Delta r_5[15] = 0$, and $\Delta r_5[14] = \Delta r_5[6] = 0$, $\Delta r_5[4] = 0$, we obtain $\Delta r_5[13] = \Delta r_5[5]$. Given $\Delta r_8[5]$ and $r_8[4,7]$, we have $\Delta r_8[4,7] = 0$, so $\Delta r_8[4,11]$ and $\Delta r_7[1,3,4,9,11,12]$ are known, allowing computation of $\Delta r_7[1,3,4,9,11,13]$. From $r_7[1,3,4,9,11,13]$ and $r_7[4,11]$, we can compute $r_7[1,3,4,9,11,12]$ and $r_6[1,3,4,9,11,12]$.

Since $r_5[0 \sim 6, 8 \sim 13, 15]$ and $r_6[0 \sim 6, 8 \sim 13, 15]$ are known, we can compute $r_5[0,5,10,15]$ and $r_6[0,1,2,3]$ using Proposition 1. From $r_8[1,3,4,9,11,12]$ and $r_7[4,11]$, we can compute $r_7[4,11]$ and $r_6[1,3,4,9,11,12]$. Given $r_5[0 \sim 5, 7 \sim 13, 15]$ and $r_6[0 \sim 6, 8 \sim 13, 15]$, we can compute $r_5[0,1,2,3,7,8,9,10,14]$ and $r_5[0,1,2,3,4,11,12,13,15]$.

Because $r_6[9] \oplus r_6[10] = r_5[9] \oplus r_5[10]$ from the MixColumn operation, and $r_6[9,10]$ can be derived from $r_5[9,10]$, we can compute $r_6[10]$. According to the key schedule, $r_2[9,10]$ can be derived from $r_6[9,10]$, and knowing $r_3[9,10]$ allows computation of $r_2[9,10]$. From the MixColumn operation, $r_2[9] \oplus r_2[10] = r_2[9] \oplus r_2[10]$, so for 2^8 values of $r_2[9]$, we can compute $r_2[10]$ and derive $r_2[3,12]$. Using the key schedule, $r_3[1,8]$ can be derived from $r_5[1,8]$, $r_4[0,1,8,9,14]$ from $r_4[0,1,8,9,14]$, and $r_7[4,11]$ from $r_5[4,11]$. Through key-dependent sieve techniques, the 46 bytes can be reduced to 31 bytes, yielding 2^{136} possible values.

3.2 10-Round Midori128 Attack Procedure

Based on the 7-round distinguisher, we construct a 10-round attack by adding one round before and two rounds after the distinguisher, as illustrated in [Figure

3: see original paper]. The attack consists of two phases: precomputation and key filtering.

1) Precomputation Phase

We construct table T_0 to store all ordered sequences $(\Delta r_{18} \oplus r_{18}^0, \cdots, \Delta r_{18} \oplus r_{18}^0)$. Before building T_0, we first construct precomputation tables T_i $(1 \leq i \leq 6)$:

- **Table T_1:** Guess all 2^{72} possible values of $\Delta r_8[5] \parallel r_8[4,11] \parallel r_7[1,3,4,9,11,12]$. Compute $r_7[1,3,4,9,11,13]$, $r_7[4,11]$, $r_8[4,11]$, and obtain $rr_7[4,11] = r_7[4,11] \oplus r_8[4,11]$. Since $\Delta r_8[5] = \Delta r_8[6]$, we can compute $\Delta r_8[4,11]$ and $\Delta r_7[1,3,4,9,11,12]$, then derive $\Delta r_7[1,3,4,9,11,12]$ and $\Delta r_6[1,3,4,9,11,12]$. Store $\Delta r_6[1,3,4,9,11,12] \parallel r_7[1,3,4,9,11,12] \parallel r_8[4,11]$ in T_1, indexed by $rr_7[4,11]$, with each index corresponding to 2^{56} stored values.

- **Table T_2:** Guess all 2^{72} possible values of $\Delta r_2[3] \parallel r_3[9,10] \parallel r_4[0,2,3,9,10,11]$. Compute $r_3[1,8]$ and $r_3[1,8]$, then obtain $rr_3[1,8] = r_3[1,8] \oplus r_3[1,8]$. Since $\Delta r_2[3] = \Delta r_2[12]$, we can compute $\Delta r_3[9,10]$ and $\Delta r_4[0,2,3,9,10,11]$, then derive $\Delta r_4[0,2,3,9,10,11]$ and $\Delta r_4[0,1,6,8,9,14]$. Store $\Delta r_4[0,1,6,8,9,14] \parallel r_4[0,2,3,9,10,11] \parallel r_3[9,10]$ in T_2, indexed by $rr_3[1,8]$, with each index corresponding to 2^{56} stored values.

- **Table T_3:** Guess all 2^{112} possible values of $\Delta r_4[1,6,8,9,14] \parallel \Delta r_5[0,1,2,3] \parallel \Delta r_6[1,3,4,11,12]$. From $\Delta r_4[1,6,8,9,14]$, compute $\Delta r_5[0,5,10,15]$; from $\Delta r_5[0,1,2,3]$, compute $\Delta r_5[0,5,10,15]$ and $\Delta r_6[0,1,2,3]$; from $\Delta r_6[1,3,4,11,12]$, compute $\Delta r_6[0,1,2,3]$. Using Proposition 1, derive $r_5[0,5,10,15]$ and $r_6[0,1,2,3]$, then compute $r_5[0,1,2,3]$ and $r_5[0,1,2,3]$, obtaining $rr_5[0,1,2,3] = r_5[0,1,2,3] \oplus r_5[0,1,2,3]$. Store $r_5[0,5,10,15] \parallel r_6[0,1,2,3]$ in T_3, indexed by $rr_5[0,1,2,3] \parallel \Delta r_4[1,6,8,9,14] \parallel \Delta r_6[1,3,4,11,12]$, with each index corresponding to one stored value.

- **Table T_4:** Guess all 2^{80} possible values of $\Delta r_4[0,6,8,9] \parallel \Delta r_5[5,7] \parallel \Delta r_6[1,3,4,9]$. Since $\Delta r_5[7] = 0$, we have $\Delta r_5[5] = \Delta r_5[6]$. From $\Delta r_4[0,6,8,9]$, compute $\Delta r_5[1,4,11]$; from $\Delta r_5[5,6,7]$, compute $\Delta r_5[1,4,11]$ and $\Delta r_6[4,5,6]$; from $\Delta r_6[1,3,4,9]$, compute $\Delta r_6[4,5,6]$. Using Proposition 1, derive $r_5[1,4,11]$ and $r_6[4,5,6]$, then compute $r_5[5,6,7]$, $r_5[4]$, and $r_5[7]$, obtaining $rr_5[7] = r_5[7] \oplus r_5[7]$ and $rr_5[4] = r_5[4] \oplus r_6[4]$. Store $r_5[1,4,11] \parallel r_6[4,5,6]$ in T_4, indexed by $rr_5[7] \parallel rr_5[4] \parallel \Delta r_4[0,6,8,9] \parallel \Delta r_6[1,3,4,9]$, with each index corresponding to one stored value.

- **Table T_5:** Guess all 2^{96} possible values of $\Delta r_4[0,1,8,14] \parallel \Delta r_5[8,9,10] \parallel \Delta r_6[3,4,9,11,12]$. From $\Delta r_4[0,1,8,14]$, compute $\Delta r_5[3,9,12]$; from $\Delta r_5[8,9,10]$, compute $\Delta r_5[3,9,12]$ and $\Delta r_6[8,9,10,11]$; from $\Delta r_6[3,4,9,11,12]$, compute $\Delta r_6[8,9,10,11]$. Using Proposition 1, derive $r_5[3,9,12]$ and $r_6[8,9,10,11]$, then compute $r_5[8,9,10]$ and

$r_5[8,9,10,11]$, obtaining $rr_5[8,9,10] = r_5[8,9,10] \oplus r_5[8,9,10]$. Store $r_5[3,9,12] \parallel r_6[8,9,10,11]$ in T_5, indexed by $rr_5[8,9,10] \parallel \Delta r_4[0,1,8,14] \parallel \Delta r_6[3,4,9,11,12]$, with each index corresponding to one stored value.

- **Table T_6:** Guess all 2^{80} possible values of $\Delta r_4[0,1,6,9,14] \parallel \Delta r_5[12,13,14] \parallel \Delta r_6[1,9,11,12]$. Since $\Delta r_5[14] = 0$, we have $\Delta r_5[15] = \Delta r_5[12] \oplus \Delta r_5[13]$. From $\Delta r_4[0,1,6,9,14]$, compute $\Delta r_5[2,7,8,13]$; from $\Delta r_5[12,13,14]$, compute $\Delta r_5[2,7,8,13]$ and $\Delta r_6[12,13,15]$; from $\Delta r_6[1,9,11,12]$, compute $\Delta r_6[12,13,15]$. Using Proposition 1, derive $r_5[2,7,8,13]$ and $r_6[12,13,15]$, then compute $r_5[12,13,15]$, obtaining $rr_5[12,13,15] = r_5[12,13,15] \oplus r_6[12,13,15]$. Store $r_5[2,7,8,13] \parallel r_6[12,13,15]$ in T_6, indexed by $rr_5[12,13,15] \parallel \Delta r_4[0,1,6,9,14] \parallel \Delta r_6[1,9,11,12]$, with each index corresponding to one stored value.

- **Table T_0:** Guess all 2^{96} possible values of $rr_5[0,1,2,3,7,8,9,10] \parallel rr_5[4,12,13,15]$. Using the key schedule, compute $rr_7[4,11]$ from $rr_4[4]$ and $rr_5[8,9,10]$. Then search T_1 using $rr_7[4,11]$ to obtain $\Delta r_6[1,3,4,9,11,12] \parallel r_7[1,3,4,9,11,12] \parallel r_8[4,11]$. Similarly, using $rr_3[1,8]$, search T_2 to obtain $\Delta r_4[0,1,6,8,9,14] \parallel r_3[9,10] \parallel r_4[0,2,3,9,10,11]$.

For each of the 2^{112} values $\Delta r_6[1,3,4,9,11,12] \parallel r_7[1,3,4,9,11,12] \parallel r_8[4,11] \parallel \Delta r_4[0,1,6,8,9,14] \parallel r_3[9,10] \parallel r_4[0,2,3,9,10,11]$, check T_3 to obtain $r_5[0,5,10,15] \parallel r_6[0,1,2,3]$. Since $rr_5[7]$ and $rr_5[4]$ are known, check T_4 using $rr_5[7] \parallel rr_5[4] \parallel \Delta r_4[0,6,8,9] \parallel \Delta r_6[1,3,4,9]$ to obtain $r_5[1,4,11] \parallel r_6[4,5,6]$. Similarly, check T_5 using $rr_5[8,9,10] \parallel \Delta r_4[0,1,8,14] \parallel \Delta r_6[3,4,9,11,12]$ to obtain $r_5[3,9,12] \parallel r_6[8,9,10,11]$. Check T_6 using $rr_5[12,13,15] \parallel \Delta r_4[0,1,6,9,14] \parallel \Delta r_6[1,9,11,12]$ to obtain $r_5[7,13,2,8] \parallel r_6[12,13,15]$.

From the values retrieved from T_3, T_4, T_5, T_6, we can compute $r_4[0,1,6,8,9,14]$ and $\overline{r}_4[0,1,6,8,9,14]$, obtaining $rr_4[0,1,6,8,9,14] = \overline{r}_4[0,1,6,8,9,14] \oplus r_4[0,1,6,8,9,14]$. Similarly, from $r_6[0 \sim 6, 8 \sim 13, 15] \parallel r_7[1,3,4,9,11,12]$, we compute $rr_6[1,3,4,9,11,12]$. Using the key schedule, compute keys $rr_5[0,1,8,9,14]$ and $rr_5[1,3,4,11,12]$ from $rr_4[0,1,8,9,14]$ and $rr_6[1,3,4,11,12]$. Compare these computed values with the originally guessed key $rr_5[0,1,8,9,14] \parallel rr_5[1,3,4,11,12]$. If they match, retain the series of values $r_3[9,10] \parallel r_4[0,2,3,9,10,11] \parallel r_5[j](j \neq 6,14) \parallel r_6[j](j \neq 7,14) \parallel rr_6[1,3,4,9,11,13] \parallel rr_7[4,11]$. There are 2^{112} retrieved values, but each is retained with probability 2^{-80}, leaving 2^{32} values.

For these 2^{32} values, derive $rr_2[9,10]$ from the key schedule and compute $r_2[9,10]$. From the MixColumn operation, $r_2[9] \oplus r_2[10] = r_2[9] \oplus r_2[10]$, so for 2^8 values of $r_2[9]$, we can compute $r_2[10]$ and derive $r_2[3,12]$. This yields all 46 bytes of parameters. Using these parameters, compute the ordered sequence $(\Delta r_{18} \oplus r_{18}^0, \cdots, \Delta r_{18} \oplus r_{18}^0)$ and store it along with the 56-bit key value $rr_5[0,1,2,3,12,13] \parallel rr_6[9]$.

2) Key Filtering Phase

Before key filtering, we precompute four tables T_i ($7 \le i \le 10$):

- **Table T_7:** Guess all 2^{72} possible values of $\Delta r_9[5] \parallel \Delta r_9[2,11] \parallel r_{10}[0,1,3,8,9,10]$. Compute $\Delta r_9[5,6]$ and $\overline{r}_9[2,11]$. Since $\Delta r_9[5] = \Delta r_9[6]$, Proposition 1 yields $r_9[5,6]$ and $r_9[2,11]$. Then compute $rr_9[2,11] = \overline{r}_9[2,11] \oplus r_9[2,11]$. Store $r_9[5,6] \parallel \Delta r_9[5]$ in T_7, indexed by $rr_9[2,11] \parallel r_{10}[0,1,3,8,9,10] \parallel \Delta r_9[2,11]$, with each index corresponding to 2^{-8} stored values.
- **Table T_8:** Guess all 2^{56} possible values of $\Delta P[0,5,10] \parallel P[0,5,10] \parallel \Delta r_1[3]$. Compute $\Delta r_1[0,5,10]$ and $\overline{r}_1[0,5,10]$. Proposition 1 gives $r_1[0,5,10]$ and $r_1[0,5,10]$, then compute $rr_0[0,5,10] = P[0,5,10] \oplus r_1[0,5,10]$ and $r_1[3]$. Store $r_1[3] \parallel \Delta r_1[3] \parallel rr_0[5]$ in T_8, indexed by $rr_0[0,10] \parallel \Delta P[0,5,10] \parallel P[0,5,10]$, with each index corresponding to 2^{-8} stored values.
- **Table T_9:** Guess all 2^{56} possible values of $\Delta P[2,8,13] \parallel P[2,8,13] \parallel \Delta r_1[12]$. Compute $\Delta r_1[2,8,13]$ and $\overline{r}_1[2,8,13]$. Proposition 1 gives $r_1[2,8,13]$ and $r_1[2,8,13]$, then compute $rr_0[2,8,13] = P[2,8,13] \oplus r_1[2,8,13]$ and $r_1[12]$. Store $r_1[12] \parallel \Delta r_1[12] \parallel rr_0[2,13]$ in T_9, indexed by $rr_0[8] \parallel \Delta P[2,8,13] \parallel P[2,8,13]$, with each index corresponding to one stored value.
- **Table T_{10}:** Guess all 2^{40} possible values of $\Delta r_1[3,12] \parallel r_1[3,12] \parallel \Delta r_2[3]$. Compute $\Delta r_2[3,12]$. Since $\Delta r_2[3] = \Delta r_2[12]$, Proposition 1 yields $r_2[3,12]$ and $r_2[3,12]$, then compute $rr_1[3,12] = r_1[3,12] \oplus r_2[3,12]$. Store $r_2[3,12] \parallel rr_1[12]$ in T_{10}, indexed by $rr_1[3] \parallel \Delta r_1[3,12] \parallel r_1[3,12]$, with each index corresponding to one stored value.

In the key filtering phase, we first find a correct pair satisfying the truncated differential chain, then compute the ordered sequence and verify it against T_0, and finally use the stored key in T_0 to validate the computed key $rr_5[0,1,2,3,12,13] \parallel rr_6[9]$.

Assume plaintext bytes $P[0,2,5,8,10,13]$ take arbitrary values while the remaining 10 bytes are fixed, forming a structure with 2^{48} plaintexts that yields $2^{48+47} = 2^{95}$ plaintext-ciphertext pairs. By selecting 2^{65} different values for $P[1,3,4,6,7,9,11,12,14,15]$, we obtain 2^{113} plaintexts forming 2^{160} pairs. Since the probability of a pair satisfying the 7-round truncated differential is $2^{(1-6+1-16)\times 8} = 2^{-160}$, on average one pair satisfies the differential under each key guess. Encrypt the 2^{113} plaintexts and filter ciphertext pairs where $\Delta C[2,4,5,6,7,11,12,13,14,15] = 0$, leaving $2^{160-10\times 8} = 2^{80}$ pairs.

For these 2^{80} pairs, perform the following:

a) Guess all 2^{16} possible values of $\Delta r_9[2,11]$ to compute $\Delta r_{10}[0,1,3,8,9,10]$. Using Proposition 1 with $\Delta r_{10}[0,1,3,8,9,10]$ and $\Delta C[0,1,3,8,9,10]$, obtain $r_{10}[0,1,3,8,9,10]$ and compute $rr_{10}[0,1,3,8,9,10]$, yielding $rr_9[2,11]$. This produces 2^{16} values of $\Delta r_9[2,11] \parallel rr_{10}[0,1,3,8,9,10] \parallel$

$rr_9[2,11] \parallel r_{10}[0,1,3,8,9,10]$. Searching T_7 yields 2^{-8} retrieved values $r_9[5,6] \parallel \Delta r_9[5]$, leaving 2^8 values that satisfy the backend differential.

b) For these 2^8 values, use the key schedule to compute $rr_0[0,10]$. Search T_8 using $rr_0[0,10] \parallel \Delta P[0,5,10] \parallel P[0,5,10]$, retrieving on average 2^{-8} values $r_1[3] \parallel \Delta r_1[3] \parallel rr_0[5]$ per query. Thus, on average one value $rr_{10}[0,1,3,8,9,10] \parallel rr_9[2,11] \parallel r_1[3] \parallel \Delta r_1[3] \parallel rr_0[5]$ is retained.

c) For this value, compute $rr_0[8]$ using the key schedule and search T_9, retrieving on average one value $r_1[12] \parallel \Delta r_1[12] \parallel rr_0[2,13]$.

d) From the obtained values, compute $rr_1[3]$ from $rr_{10}[3]$ and search T_{10}, retrieving one value $r_2[3,12] \parallel rr_1[12]$.

For each plaintext-ciphertext pair, on average only one possible value $rr_{10}[0,1,3,8,9,10] \parallel rr_9[2,11] \parallel rr_0[0,2,5,8,10,13] \parallel rr_1[3,12] \parallel r_2[3,12]$ remains.

e) Vary $r_2[3,12]$ based on the ordered sequence differences to obtain a set $\{r_2^0, r_2^1, \cdots, r_2^{18}\}$. Derive the corresponding plaintexts $\{P^0, P^1, \cdots, P^{18}\}$ and compute their ciphertexts $\{C^0, C^1, \cdots, C^{18}\}$. Decrypt to obtain the sequence $(\Delta r_{18} \oplus r_{18}^0, \cdots, \Delta r_{18} \oplus r_{18}^0)$. If the computed sequence is not in T_0, eliminate the key. If it exists in T_0, further verify whether the key $rr_0[2,13] \parallel rr_1[3,12] \parallel rr_{10}[0,1,3,9]$ can derive $rr_5[0,1,2,3,12,13] \parallel rr_6[9]$ from T_0. If so, retain the key with probability $2^{(17-18)\times 8-56} = 2^{-64}$.

Finally, perform exhaustive search on the remaining 2^{16} keys and 6 unguessed key bytes.

3.3 Complexity Analysis

The complexity analysis considers both precomputation and key filtering phases.

Precomputation Phase: Constructing T_0 requires 2^{136} partial encryptions of 7 rounds for 19 message values, yielding a time complexity of $2^{136} \times 19 \times 44/160 = 2^{138}$ 10-round Midori128 encryptions. The memory complexity is $2^{136} \times 19 \times 8 \times 2^{-7} = 2^{136}$ 128-bit blocks, which dominates this phase.

Key Filtering Phase: Encrypting 2^{113} chosen plaintexts requires 2^{113} 10-round encryptions, giving a data complexity of 2^{113} plaintexts. Filtering the remaining 2^{80} pairs and partially encrypting 19 message values to verify the ordered sequence requires $2^{80} \times 19 \times 16/160 = 2^{81}$ 10-round encryptions. Thus, the key filtering phase has data complexity 2^{113} chosen plaintexts, time complexity 2^{81} encryptions, and memory complexity 2^{82} 128-bit blocks.

Overall Complexity: The total attack requires 2^{113} chosen plaintexts, 2^{138} 10-round encryptions, and 2^{136} 128-bit blocks.

Optimization: To optimize complexity, we apply a time-memory tradeoff with factor $\alpha = 2^{12}$. The precomputation phase time complexity becomes 2^{126} encryptions and memory complexity 2^{124} 128-bit blocks. The key filtering phase

data complexity becomes 2^{125} chosen plaintexts, time complexity $2^{125+93} \approx 2^{125}$ encryptions, and memory complexity 2^{94} 128-bit blocks. Therefore, the overall attack achieves time complexity $2^{126} + 2^{125} \approx 2^{126.5}$ 10-round encryptions, data complexity 2^{125} plaintexts, and memory complexity $2^{124} + 2^{94} \approx 2^{124}$ 128-bit blocks.

Weak Key Attack: Since the guessed keys in precomputation and filtering phases include $ru_5[0,1,2,3]$ and $rk_{10}[0,1,3] \parallel rk_0[2]$, we can exploit their relationship through the key schedule for a weak key attack. For each fixed value of $ru_5[0,1,2,3]$, we build a subtable T_0^*, reducing the storage complexity by a factor of 2^{-32}. The memory complexity for T_0^* becomes negligible in precomputation, while the precomputation tables dominate, resulting in overall storage complexity of 2^{105} 128-bit blocks.

4 Conclusion

This paper investigates the security of Midori128 against meet-in-the-middle attacks. By employing differential enumeration and key-dependent sieve techniques, we construct a 7-round meet-in-the-middle distinguisher and extend it to mount the first known attack on 10-round Midori128. Using time-memory tradeoffs and weak key techniques, we reduce the attack complexities to 2^{125} chosen plaintexts, $2^{126.5}$ 10-round encryptions, and 2^{105} 128-bit blocks. Our analysis demonstrates that 10-round Midori128 does not provide strong resistance against meet-in-the-middle attacks, while higher-round variants currently remain secure against this attack vector.

References

[1] Banik S, Bogdanov A, Isobe T, et al. Midori: a block cipher for low energy [C]// Advances in Cryptology-ASIACRYPT 2015. Berlin: Springer, 2015.

[2] Dong Xiaoyang, Shen Yanzhao. Cryptanalysis of reduced-round Midori64 block cipher [EB/OL]. http://eprint.iacr.org/2016/676.pdf.

[3] Chen Zhan, Bi Wenquan, Wang Xiaoyun. Impossible differential cryptanalysis of Midori64 [EB/OL]. http://eprint.iacr.org/2016/535.pdf.

[4] Takahashi Y. Higher-order differential attack on the round-reduced variants of the block cipher Midori64, IEICE Tech. Rep. [R]. 2016: 159-164.

[5] Lin Li, Wu Wenling. Meet-in-the-middle attacks on reduced-round Midori-64 [EB/OL]. http://eprint.iacr.org/2015/1165.pdf.

[6] Guo Jian, Jean J, Nikolić I, et al. Invariant subspace attack against full Midori64 [EB/OL]. http://eprint.iacr.org/2015/1189.pdf.

[7] Chen Zhan, Chen Huaifeng, Wang Xiaoyun. Cryptanalysis of Midori128 using impossible differential techniques [C]// Proc of Information Security Practice and Experience. Berlin: Springer, 2016: 1-12.

[8] Bi Wenquan, Li Zheng, Dong Xiaoyang. Impossible differential attack on Midori128 using rebound-like technique [EB/OL]. http://eprint.iacr.org/2017/286.pdf.

[9] Tolba M, Abdelkhalek A, Youssef A M. Truncated and multiple differential cryptanalysis of reduced round Midori128 [C]// Proc of Information Security. Berlin: Springer, 2016: 3-17.

[10] Gérault D, Lafourcade P. Related-key cryptanalysis of Midori [C]// Proc of Cryptology-INDOCRYPT 2016. Berlin: Springer, 2016: 287-304.

[11] Li Rongjia, Jin Chenhui. Meet-in-the-middle attacks on 10-round AES-256 [J]. Designs, Codes and Cryptography, 2016, 80(3): 459-471.

[12] Diffie W, Hellman M E. Special feature exhaustive cryptanalysis of the NBS data encryption standard [J]. Computer, 1977, 10(6): 74-84.

[13] Li Leibo, Jia K, Wang Xiaoyun, et al. Improved meet-in-the-middle attacks on AES-192 and PRINCE [EB/OL]. http://eprint.iacr.org/2013/573.pdf.

[14] Daemen J, Rijmen V. The design of Rijndael: AES-the advanced encryption standard [M]. Berlin: Springer, 2002.

[15] Gilbert H, Minier M. A collision attack on 7 rounds of Rijndael [C]// Proc of the 3rd AES Candidate Conference. Berlin: Springer, 2000: 230-241.

[16] Demirci H, Selçuk A A. A meet-in-the-middle attack on 8-round AES [C]// Proc of Fast Software Encryption. Berlin: Springer, 2008: 116–126.

[17] Dunkelman O, Keller N, Shamir A. Improved single-key attacks on 8-round AES-192 and AES-256 [C]// Proc of Advances in Cryptology-ASIACRYPT 2010. Berlin: Springer, 2010: 158-176.

[18] Li Leibo, Jia K, Wang Xiaoyun. Improved single-key attacks on 9-round AES-192/256 [C]// Proc of Fast Software Encryption. Berlin: Springer, 2015.

[19] Guo Jian, Peyrin T, Poschmann A, et al. The LED block cipher [C]// Proc of Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2011.

[20] Li Leibo, Jia K, Wang Xiaoyun, et al. Meet-in-the-middle technique for truncated differential and its applications to CLEFIA and Camellia [C]// Proc of Fast Software Encryption. Berlin: Springer, 2015: 48-70.

[21] Wang Yanfeng, Wu Wenling. Meet-in-the-middle attacks on block cipher TWINE [J]. Journal of Software, 2015, 26(10): 2684-2695.

[22] Zheng Yafei, Wu Wenling. Improved meet-in-the-middle attack on LBlock algorithm [J]. Chinese Journal of Computers, 2017, 40(5): 1080-1091.

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv —Machine translation. Verify with original.