IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2016, No. 1, pp. 33–56. DOI:10.13154/tosc.v2016.i1.33-56

Invariant Subspace Attack Against Midori64 and The Resistance Criteria for S-box Designs

Jian Guo[1], Jérémy Jean[1,2], Ivica Nikolić[1],
Kexin Qiao[3,1], Yu Sasaki[1,4] and Siang Meng Sim[1]

[1] Nanyang Technological University, Singapore, Singapore
ntu.guo@gmail.com, jean.jeremy@gmail.com, inikolic@ntu.edu.sg, crypto.s.m.sim@gmail.com
[2] Agence nationale de la sécurité des systèmes d'information (ANSSI), Paris, France
[3] State Key Laboratory of Information Security (SKLOIS), Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
qiaokexin@iie.ac.cn
[4] NTT Secure Platform Laboratories, Tokyo, Japan
sasaki.yu@lab.ntt.co.jp

Abstract. We present an invariant subspace attack on the block cipher Midori64, proposed at Asiacrypt 2015. Our analysis shows that Midori64 has a class of 2^{32} weak keys. Under any such key, the cipher can be distinguished with only a single chosen query, and the key can be recovered in 2^{16} time with two chosen queries. As both the distinguisher and the key recovery have very low complexities, we confirm our analysis by implementing the attacks.
Some tweaks of round constants make Midori64 more resistant to the attacks, but some lead to even larger weak-key classes. To eliminate the dependency on the round constants, we investigate alternative S-boxes for Midori64 that provide certain level of security against the found invariant subspace attacks, regardless of the choice of the round constants. Our search for S-boxes is enhanced with a dedicated tool which evaluates the depth of any given 4-bit S-box that satisfies certain design criteria. The tool may be of independent interest to future S-box designs.

Keywords: Midori · Block Cipher · Invariant Subspace Attack · Weak Key

1 Introduction

Designing a block cipher simultaneously achieving high efficiency and high security has been a challenging topic for many years. Dozens of lightweight ciphers have been proposed for in last decade, and it is important to select good designs.

Regarding efficiency, several evaluation criteria can be considered, such as gate size, throughput and latency. One of the most important criteria is a low energy consumption. For example, in a sensor network, many sensor nodes having limited amount of computation resources and battery will be distributed. Substituting old nodes with empty battery into new ones requires an expensive maintenance cost, thus making the amount of energy consumption as low as possible is crucial in such a situation.

Midori is a family of lightweight block ciphers published at Asiacrypt 2015 [BBI+15], which have been advertised as one of the first lightweight ciphers optimized with respect to energy consumed by the circuit per bit in encryption or decryption operation. To achieve the desired low energy goal, several design decisions were made in Midori. It adopts an AES-like SPN structure, and the diffusion layer consists of almost MDS 4×4 binary matrices. The 4-bit S-box has a small delay, i.e. 1.5-2 times faster than those of

Licensed under Creative Commons License CC-BY 4.0.
Received: 2016-06-01, Accepted: 2016-08-01, Published: 2016-12-01

PRINCE [BCG+12] and PRESENT [BKL+07]. The round constants are seemingly random binary values extracted for the constant π. The key schedule is trivial and efficient. Finally, the number of rounds is rather small in comparison to other lightweight ciphers: only 16-20 rounds are used.

Given the above choices of round function operations, security of the design must be carefully discussed. The submission document of Midori contains a standard analysis of the proposed ciphers against various types of attacks: differential and linear, boomerangs, impossible differentials, etc. As a result, it has been concluded that the ciphers provide a safe security margin. Additional analysis of Midori has been provided in [LW15], which shows that 12 rounds (out of 16) of Midori64 can be attacked with the meet-in-the-middle technique, with a rather high complexity: the key recovery requires around 2^{55} chosen plaintexts, 2^{106} memory, and $2^{125.5}$ computations.

Our Contributions. We show that Midori64 has a class of 2^{32} weak keys that can be distinguished with a complexity of a single query. Furthermore, within this class of keys, a key recovery can be efficiently achieved given two plaintext-ciphertext pairs, including the pair used in the distinguisher.

Our analysis is based on the invariant subspace attacks [LAAZ11]. It uses the unfortunate combination in Midori64 of round constants, S-box, and multiplication by binary matrix in the diffusion layer. When each cell of the master key has the value 0 or 1 (in total 2^{32} such keys), and each cell of the state (including the plaintext) has the value of 8 or 9, then the transformations in Midori64 keep the state in the same class (of cells values 8 and 9). Hence, regardless of the number of rounds, the class is maintained, and as a result, the ciphertext belongs to this class as well. This fact allows to launch an efficient distinguisher. The key recovery uses an additional fact: the values 8 and 9 are fixed points for the S-box used in Midori64. As a result, the whole cipher under the weak-key class becomes a linear transformation (the only non-linear component, the S-box, turns into the identity mapping). Therefore, recovering the key is equivalent to solving a system of linear equations and it can be achieved given only two pairs of plaintext-ciphertext verifying the distinguisher. We have confirmed the correctness of the whole analysis by implementing independently the distinguisher and the key recovery. At the current stage, our attacks do not apply to Midori128.

Our attacks can be prevented with a change of the round constants of Midori64. On the other hand, there exist such constants that allow even larger weak-key classes. Hence, in the second part of the paper we analyze S-box alternatives that provide security against invariant subspace attacks, regardless of the round constants. That is, we examine the possibility to show that a cipher is resistant against invariant subspace attacks (or has only a small set of weak keys) by focusing only on the S-boxes in combination with the key schedule. We stress that we are not encouraging to remove round constants, but to find strong S-boxes that resist the attack free to choose any round constants without worries. Note, we take into account only invariant subspace attacks as our attack on Midori64, however, Leander et al. [LMR15] have shown several other flavours of these attacks. We show that involution S-boxes can provide certain level of security (without analyzing the round constants) only when the key schedule is very simple (consists of identical round keys). On the other hand, with non-involution S-boxes we can extend the security to arbitrary key schedules. In both of the cases, we provide actual S-boxes found with a dedicated tool that covers the space of all 4-bit S-boxes with certain design criteria (good linear and differential properties and low number of fixed points, besides and low depth).

2 Preliminaries

2.1 Description of Midori

Midori consists of two algorithms Midori64 and Midori128. The block size, n, is 64 bits for Midori64 and 128 bits for Midori128, and the key size is 128 bits for both. The ciphers adopt a standard SPN structure, and the internal state is represented as 4×4 cells, where the size of each cell is 4 bits for Midori64 and 8 bits for Midori128. The state S has sixteen cells $s_0, s_1, \ldots, s_{15}$ arranged as :

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

From the 128-bit master key, Midori64 and Midori128 generate a 64-bit whitening key WK and $r-1$ 64-bit round keys $RK_0, RK_1, \ldots, RK_{r-2}$. Here, r is the number of rounds, which is 16 for Midori64 and 20 for Midori128. The plaintext is first loaded into the state and the whitening key WK is XORed to the state. Then, the round function $RF : \{0,1\}^n \times \{0,1\}^{64} \mapsto \{0,1\}^n$, which takes as input the current state and the round key RK_i and outputs the updated state, is iterated $r-1$ times. Finally, the last round function RF^l is applied and the resulting state is output as the ciphertext.

2.1.1 Key Generation

In Midori64, the 128-bit key K is separated into two 64-bit states K_0 and K_1. Then, the whitening key WK is computed as $K_0 \oplus K_1$, and the round keys RK_i for $i = 0, 1, \ldots 14$ are computed as $K_{(i \bmod 2)} \oplus \alpha_i$, where α_i is a round constant described below.

The round constants α_i where $i = 0, 1, \ldots, 14$ consist of 16 binary cells. The constants have been derived from the hexadecimal encoding of the fractional part of π. For example, α_0 and α_1 are defined as follows:

$$\alpha_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \text{and} \quad \alpha_1 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

The remaining α_i are defined similarly. Refer to [BBI+15] for more details. We later exploit the fact that all the α_i's are binary matrices, i.e. all the cells in any α_i are either 0 or 1.

2.1.2 Round Function and the Last Round Function

The round function RF consists of the four operations SubCell, ShuffleCell, MixColumn and KeyAdd that update the n-bit state S.

SubCell This operation in Midori64 applies a 4-bit S-box Sb_0 to each cell, while in Midori128 applies four 8-bit S-boxes SSb_0, SSb_1, SSb_2 and SSb_3 to each of the four cells in Row 0, Row 1, Row 2 and Row 3, respectively. Each SSb_i is generated by 4-bit S-box Sb_1. Refer to [BBI+15] for the details of how to generate SSb_i from Sb_1. The full specifications of Sb_0 and Sb_1 are shown in Table 1.

Table 1: Specifications of Sb_0 and Sb_1

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathsf{Sb}_0(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6
$\mathsf{Sb}_1(x)$	1	0	5	3	e	2	f	7	d	a	9	b	c	8	4	6

ShuffleCell This transformation is a cell-wise permutation. Each cell is permuted as follows.

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

MixColumn This transformation applies a 4×4 binary involution matrix to each column of the state as follows.

$$\begin{pmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ s_{i+3} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ s_{i+3} \end{pmatrix}, \text{ for } i \in \{0, 4, 8, 12\}.$$

KeyAdd $\mathsf{KeyAdd}(S, RK_i)$ cell-wise XORs RK_i to the state S.

The last round function RF^l only applies two operations; namely, $\mathsf{SubCell}(S)$ and $\mathsf{KeyAdd}(S, WK)$.

Algorithm 1 – Midori encryption algorithm

```
function MIDORI-ENCRYPTION(P)
    S ← P
    S ← KeyAdd(S, WK)
    for i = 0, ..., r − 2 do
        S ← SubCell(S)
        S ← ShuffleCell(S)
        S ← MixColumn(S)
        S ← KeyAdd(S, RK_i)
    end for
    S ← SubCell(S)
    S ← KeyAdd(S, WK)
    return S
end function
```

2.1.3 Summary

The encryption of Midori can be summarized as shown in Algorithm 1, and Midori64 encryption function is depicted in Figure 1. Note that the decryption can be described similarly. However, since our attack only uses the encryption, we omit the description of the decryption.

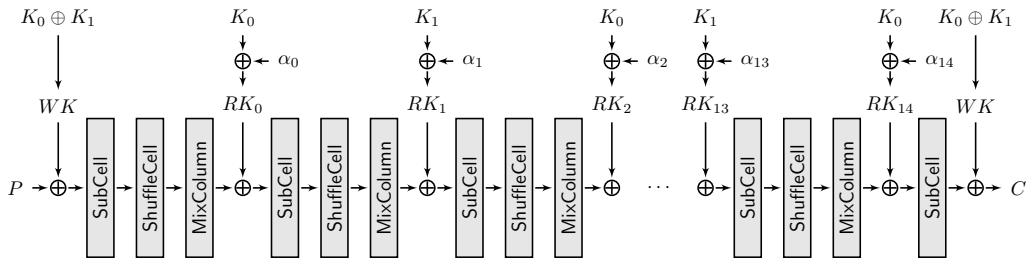

Figure 1: Midori64 encryption algorithm.

2.2 The Invariant Subspace Attack

As a method of cryptanalysis, the invariant subspace attack was introduced by Leander et al. at CRYPTO 2011 [LAAZ11]. In this method, the adversary aims to find so-called invariant subspaces, i.e. subsets of the set of all possible state and key values, invariant of the round transformations used in the analyzed cipher. When such a subset exists, then the adversary encrypts plaintexts that belong to the subset, assumes the master key belongs as well to the subset (thus it is a weak-key attack) and expects to obtain corresponding ciphertexts that also belong to the subset. This immediately yields a distinguisher for the cipher, while more advanced approaches can be used for a key recovery.

The invariant subspaces for an n-bit iterated cipher with a round function $F_{K_i}(x) = R(x \oplus K_i)$, where x is the state, K_i is the subkey of round i, formally can be introduced as follows. Assume there exist two constants $u, v \in F_2^n$ and a subspace $A \subseteq F_2^n$ such that

$$R(u \oplus A) = v \oplus A.$$

If all the subkeys K_i are such that $K_i \in u \oplus v \oplus A$, then it follows:

$$F_{K_i}(v \oplus A) = R(u \oplus v \oplus A \oplus v \oplus A) = R(u \oplus A) = v \oplus A.$$

Therefore, if the plaintext $P \in v \oplus A$, it follows that the ciphertext $C \in v \oplus A$ regardless of the number of rounds.

Non-trivial invariant subspaces do not necessarily exist for a given cipher. When they do exist, they are found either by a careful analysis (as it is the case of the analysis of PRINTCipher [LAAZ11]) or with the use of a specialized tool [LMR15]. To deduce the invariant subspace, the former method requires examination of all the transformations used in the cipher, which usually provides a hint of the possible subspace. On the other hand, the latter method is generic and is achieved by a computer search. Its feasibility depends on the proportion of the sizes of weak to all key class.

3 Invariant Subspace Attack on Midori64

In this section, we present the invariant subspace attack on Midori64. Our analysis reveals a class of 2^{32} weak keys. Within this class, Midori64 can be distinguished from a random permutation with a single chosen plaintext query, a negligible computational cost, and a negligible memory. Moreover, the key can be recovered from the 2^{32} potential candidates in 2^{16} operations.

3.1 Distinguisher with Invariant Subspace Attack

We first introduce several notations used in this attack.

$\mathcal{K}$: a subspace of cell values consisting of two elements 0 and 1, i.e., $\mathcal{K} \triangleq \{0, 1\}$

$\mathbb{K}$: a subspace of state values in which each of its sixteen cells belongs to $\mathcal{K}$, i.e., $\mathbb{K} \triangleq \mathcal{K}^{16}$

$\mathcal{S}$: an affine subspace of cell values consisting of two elements 8 and 9, i.e., $\mathcal{S} \triangleq \{8, 9\} = 8 \oplus \mathcal{K}$

$\mathbb{S}$: an affine subspace of state values in which each of its sixteen cells belongs to $\mathcal{S}$, i.e., $\mathbb{S} \triangleq \mathcal{S}^{16}$

Proposition 1 (Invariant Subspace)**.** *If the 128-bit master key $K_0 \| K_1$ satisfies $K_0, K_1 \in \mathbb{K}$, then any plaintext $P \in \mathbb{S}$ is mapped by Midori64 to a ciphertext $C \in \mathbb{S}$ with probability one.*

Throughout this section, we prove Proposition Proposition 1. To achieve this, we focus independently on each transformation used in Midori64.

3.1.1 Round Key Generation

Let $x, y \in \mathcal{K}$. Then, $x \oplus y \in \mathcal{K}$. Therefore, for any $X, Y \in \mathbb{K}$, $X \oplus Y \in \mathbb{K}$.

The whitening key WK is computed by $K_0 \oplus K_1$. By assuming $K_0, K_1 \in \mathbb{K}$, we have $WK \in \mathbb{K}$.

The round key for the i-th round, RK_i, is computed by $K_{(i \bmod 2)} \oplus \alpha_i$. Here, an important observation for our attack is that all the round constants α_i only consist of 0 and 1, i.e., $\alpha_i \in \mathbb{K}$ for $i = 0, 1, \ldots, 14$. By assuming $K_0, K_1 \in \mathbb{K}$, we have $RK_i \in \mathbb{K}$ for all $i = 0, 1, \ldots, 14$.

3.1.2 Data Processing Part

Let $x \in \mathcal{S}$ and $y \in \mathcal{K}$. Then, $x \oplus y \in \mathcal{S}$. Therefore, for any $X \in \mathbb{S}$ and $Y \in \mathbb{K}$, $X \oplus Y \in \mathbb{S}$. As long as the plaintext $P \in \mathbb{S}$, the state after adding the whitening key, $WK \in \mathbb{K}$, belongs to $\mathbb{S}$.

Then, the state is processed by the SubCell operation. Here, we exploit two particular data transitions through the S-box for Midori64; $\mathsf{Sb}_0(8) = 8$ and $\mathsf{Sb}_0(9) = 9$. Namely, as long as the input state belongs to $\mathbb{S}$, SubCell is equivalent to the identity mapping. Obviously, we obtain $\mathbb{S} \leftarrow \mathsf{SubCell}(\mathbb{S})$.

The subsequent ShuffleCell is a cell-wise permutation, and since all cells in $\mathbb{S}$ satisfy $\mathcal{S}$, $\mathbb{S} \leftarrow \mathsf{ShuffleCell}(\mathbb{S})$.

The MixColumn operation is slightly more complex. Because the diffusion matrix is a binary matrix, each output cell from MixColumn can be represented as the XOR of three input cells. As long as the input state belongs to $\mathbb{S}$, each of three cells is either 8 or 9. Thus, the possibilities for each output cell is the following eight cases:

$$\begin{aligned} &8 \oplus 8 \oplus 8 = 8, && 8 \oplus 8 \oplus 9 = 9, && 8 \oplus 9 \oplus 8 = 9, && 8 \oplus 9 \oplus 9 = 8, \\ &9 \oplus 8 \oplus 8 = 9, && 9 \oplus 8 \oplus 9 = 8, && 9 \oplus 9 \oplus 8 = 8, && 9 \oplus 9 \oplus 9 = 9. \end{aligned}$$

In any case, each output cell belongs to $\mathcal{S}$, thus $\mathbb{S} \leftarrow \mathsf{MixColumn}(\mathbb{S})$.

The KeyAdd operation is the same as the whitening key addition, i.e.:

$$\mathbb{S} \leftarrow \mathsf{KeyAdd}(\mathbb{S}, RK_i \in \mathbb{K}).$$

3.1.3 Summary

Thanks to the property of $\alpha_i \in \mathbb{K}$, any weak key $K_0, K_1 \in \mathbb{K}$ leads to $WK \in \mathbb{K}$ and $RK_i \in \mathbb{K}$. Let $P \in \mathbb{S}$. Then, the state after the whitening key addition becomes

$\mathbb{S} \leftarrow \mathsf{KeyAdd}(P \in \mathbb{S}, WK \in \mathbb{K})$. Then, the following round function is iterated by incrementing the round number i.

$$\begin{aligned}
&\mathbb{S} \leftarrow \mathsf{SubCell}(\mathbb{S}),\\
&\mathbb{S} \leftarrow \mathsf{ShuffleCell}(\mathbb{S}),\\
&\mathbb{S} \leftarrow \mathsf{MixColumn}(\mathbb{S}),\\
&\mathbb{S} \leftarrow \mathsf{KeyAdd}(\mathbb{S}, RK_i \in \mathbb{K}).
\end{aligned}$$

As a result, regardless of the number of rounds applied, the state belongs to $\mathbb{S}$ with probability one. The last round consists of only SubCell and the whitening key addition, which does not break the property. This completes the proof of Proposition Proposition 1.

By following the notations in [LMR15], the affine subspace $8 \oplus \{0, 1\}$ is mapped to itself with SubCell, ShuffleCell, MixColumn and KeyAdd when $RK_i \in \mathbb{K}$.

3.1.4 Experiments

We implemented our invariant subspace distinguisher and verified its correctness. Some examples are shown in Table 2.

Table 2: Experimental data

	Example 1	**Example 2**	**Example 3**
K_0	0000000000000000	1100110011001100	0000101001001110
K_1	0000000000000000	0011001100110011	1101010100010001
P	8888888888888888	9999999999999999	9889898898898989
C	9998899889888899	8999999988988989	9999988988898889

3.1.5 Computer Search of Invariant Subspaces

We performed a computer search to detect the largest weak-key class. We brute-forced all possible subspaces of cell values in the plaintext (each cell belongs to the same subset) and all possible values of master key cells (similarly, they all belong to another subset). As there are 16 values for the cells in each of the two cases, the brute-force required around $2^{16} \cdot 2^{16} = 2^{32}$ time. We found five subspaces, all subsets of the original subspace. Thus, we can conclude that no larger weak-key classes of the *analyzed type* exist in Midori64.

We emphasize that while the generic search algorithm presented in [LMR15] could detect Midori64's invariant subspaces, about $2^{2(64-16)} = 2^{96}$ operations are required for brute-forcing all the possibilities. Even with their advanced *probabilistic* search, about $50 \times 2^{64-16} \approx 2^{53.6}$ operations are required. Indeed, the time complexity of this generic algorithm decreases exponentially with the dimension of the subspace, making it harder to detect small subspaces like in Midori64 (being apart from generic, reducing the search space to feasible one is possible by using the specific structure of Midori64's round function).

In contrast, the exhaustive analysis we present in the sequel has been found by careful analysis of the components of the cipher without using the generic invariant subspace detection algorithm.

3.2 Key Recovery with Invariant Subspace Attack

In this section, we describe how a chosen plaintext P and its corresponding ciphertext C satisfying the subspace distinguisher can be used to efficiently recover the 128-bit weak key. Because the size of the weak-key class is 2^{32}, the exhaustive search on the entire weak-key space requires 2^{32} computations. Hence, our goal is to recover the key in time less than 2^{32}.

The main observation pertains to the behavior of the S-box on the subset $\mathcal{S}$. Indeed, the S-box Sb_0 used in Midori64 has four fixed points $\mathcal{S} \subset \{3, 7, 8, 9\}$. Consequently, under the assumption that $S \in \mathbb{S}$, the S-box behaves like the identity mapping, which in turn makes linear the full Midori64 cipher.

Therefore, recovering the 128-bit key $K = K_0||K_1$ can be done by writing the system of linear equations between $P \in \mathbb{S}$ and $C \in \mathbb{S}$. To describe the system, we denote by $k_0, \ldots, k_{15}$ the 16 variables from K_0, and by $k_{16}, \ldots, k_{31}$ the 16 ones from K_1. We emphasize that $k_i \in \mathcal{K}$, since we assume that K belongs to the weak-key class $\mathbb{K}$. Similarly, we denote the 16 known variables of the plaintext P by $p_0, \ldots, p_{15}$ and the 16 known variables of the ciphertext C by $c_0, \ldots, c_{15}$, that is:

$$K_0 = \begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix} \in \mathbb{K}, \qquad K_1 = \begin{bmatrix} k_{16} & k_{20} & k_{24} & k_{28} \\ k_{17} & k_{21} & k_{25} & k_{29} \\ k_{18} & k_{22} & k_{26} & k_{30} \\ k_{19} & k_{23} & k_{27} & k_{31} \end{bmatrix} \in \mathbb{K},$$

$$P = \begin{bmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{bmatrix} \in \mathbb{S}, \qquad C = \begin{bmatrix} c_0 & c_4 & c_8 & c_{12} \\ c_1 & c_5 & c_9 & c_{13} \\ c_2 & c_6 & c_{10} & c_{14} \\ c_3 & c_7 & c_{11} & c_{15} \end{bmatrix} \in \mathbb{S}.$$

Under these notations, the linear system of 16 equations becomes:

$$\begin{aligned}
k_0 \oplus k_{11} \oplus k_{14} \oplus k_{15} \oplus k_{21} \oplus k_{22} \oplus k_{23} \oplus k_{26} \oplus k_{28} \oplus k_{29} \oplus k_{30} \oplus k_{31} &= p_0 \oplus p_5 \oplus p_6 \oplus p_7 \oplus p_{10} \oplus p_{11} \oplus p_{12} \oplus p_{13} \\
&\oplus c_5 \oplus c_6 \oplus c_7 \oplus c_{10} \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15} \\
k_1 \oplus k_{11} \oplus k_{19} \oplus k_{24} \oplus k_{26} \oplus k_{29} \oplus k_{31} &= p_1 \oplus p_3 \oplus p_8 \oplus p_{10} \oplus p_{11} \oplus p_{13} \oplus p_{15} \\
&\oplus c_3 \oplus c_8 \oplus c_{10} \oplus c_{13} \oplus c_{15} \oplus 1 \\
k_2 \oplus k_{14} \oplus k_{19} \oplus k_{21} \oplus k_{22} \oplus k_{23} \oplus k_{24} \oplus k_{28} \oplus k_{30} \oplus k_{31} &= p_2 \oplus p_3 \oplus p_5 \oplus p_6 \oplus p_7 \oplus p_8 \oplus p_{12} \oplus p_{15} \\
&\oplus c_3 \oplus c_5 \oplus c_6 \oplus c_7 \oplus c_8 \oplus c_{12} \oplus c_{14} \oplus c_{15} \\
k_3 \oplus k_{15} \oplus k_{19} \oplus k_{24} \oplus k_{25} \oplus k_{29} &= p_8 \oplus p_9 \oplus p_{13} \oplus p_{15} \oplus c_3 \oplus c_8 \oplus c_9 \oplus c_{13} \oplus 1 \\
k_4 \oplus k_{11} \oplus k_{13} \oplus k_{15} \oplus k_{22} \oplus k_{25} \oplus k_{27} \oplus k_{28} \oplus k_{29} \oplus k_{30} &= p_4 \oplus p_6 \oplus p_9 \oplus p_{12} \oplus p_{14} \oplus p_{15} \\
&\oplus c_6 \oplus c_9 \oplus c_{11} \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus 1 \\
k_5 \oplus k_{14} \oplus k_{22} \oplus k_{23} \oplus k_{25} \oplus k_{28} \oplus k_{29} \oplus k_{30} &= p_5 \oplus p_6 \oplus p_7 \oplus p_9 \oplus p_{12} \oplus p_{13} \\
&\oplus c_6 \oplus c_7 \oplus c_9 \oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus 1 \\
k_6 \oplus k_{13} \oplus k_{14} \oplus k_{15} \oplus k_{22} \oplus k_{25} \oplus k_{28} \oplus k_{29} &= p_9 \oplus p_{12} \oplus p_{14} \oplus p_{15} \oplus c_6 \oplus c_9 \oplus c_{12} \oplus c_{13} \\
k_7 \oplus k_{13} \oplus k_{14} \oplus k_{15} \oplus k_{23} &= p_{13} \oplus p_{14} \oplus p_{15} \oplus c_7 \\
k_8 \oplus k_{15} \oplus k_{24} \oplus k_{29} &= p_{13} \oplus p_{15} \oplus c_8 \oplus c_{13} \\
k_9 \oplus k_{11} \oplus k_{13} \oplus k_{14} \oplus k_{24} \oplus k_{28} &= p_8 \oplus p_9 \oplus p_{11} \oplus p_{12} \oplus p_{13} \oplus p_{14} \oplus c_8 \oplus c_{12} \\
k_{10} \oplus k_{11} \oplus k_{25} &= p_9 \oplus p_{10} \oplus p_{11} \oplus c_9 \oplus 1 \\
k_{12} \oplus k_{13} \oplus k_{14} \oplus k_{15} \oplus k_{29} &= p_{12} \oplus p_{14} \oplus p_{15} \oplus c_{13} \\
k_{16} \oplus k_{19} \oplus k_{24} \oplus k_{25} \oplus k_{29} \oplus k_{31} &= p_0 \oplus p_3 \oplus p_8 \oplus p_9 \oplus p_{13} \oplus p_{15} \\
&\oplus c_0 \oplus c_3 \oplus c_8 \oplus c_9 \oplus c_{13} \oplus c_{15} \oplus 1 \\
k_{17} \oplus k_{19} \oplus k_{22} \oplus k_{23} \oplus k_{24} \oplus k_{25} \oplus k_{26} \oplus k_{27} \oplus k_{28} \oplus k_{31} &= p_1 \oplus p_3 \oplus p_6 \oplus p_7 \oplus p_8 \oplus p_9 \oplus p_{10} \oplus p_{11} \\
&\oplus p_{12} \oplus p_{15} \oplus c_1 \oplus c_3 \oplus c_6 \oplus c_7 \oplus c_8 \oplus c_9 \\
&\oplus c_{10} \oplus c_{11} \oplus c_{12} \oplus c_{15} \\
k_{18} \oplus k_{19} \oplus k_{21} \oplus k_{22} \oplus k_{23} \oplus k_{24} \oplus k_{28} \oplus k_{29} \oplus k_{30} \oplus k_{31} &= p_2 \oplus p_3 \oplus p_5 \oplus p_6 \oplus p_7 \oplus p_8 \oplus p_{12} \oplus p_{13} \\
&\oplus p_{14} \oplus p_{15} \oplus c_2 \oplus c_3 \oplus c_5 \oplus c_6 \oplus c_7 \oplus c_8 \\
&\oplus c_{12} \oplus c_{13} \oplus c_{14} \oplus c_{15} \oplus 1 \\
k_{20} \oplus k_{22} \oplus k_{23} \oplus k_{25} \oplus k_{28} \oplus k_{29} &= p_4 \oplus p_6 \oplus p_7 \oplus p_9 \oplus p_{12} \oplus p_{13} \\
&\oplus c_4 \oplus c_6 \oplus c_7 \oplus c_9 \oplus c_{12} \oplus c_{13},
\end{aligned}$$

where there are 32 unknowns. The system being undetermined, the set of solution contains 2^{16} elements, which provides 2^{16} key candidates for the 128-bit master key K. Using an additional known plaintext-ciphertext pair, we uniquely determine the key in 2^{16} operations. More precisely, the above system of equations describes a Gröbner Basis so that one can simply enumerate all the 2^{16} values for $k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}, k_{12}, k_{16}, k_{17}, k_{18}, k_{20} \in \mathcal{K}$ and uniquely and efficiently determine the remaining 16 key variables.

4 Extended Analysis: Weaker Constant

The selection of the round constants (which currently have cells that are either 0 or 1) certainly has contributed towards the existence of the invariant subspace for the whole

cipher. There are, however, round constants that allow even larger invariant subspaces. We further describe such constants. Similarly to the original selection, we assume that all cells of the round constants belong to a particular set RC which is a proper subset of $\{0,1\}^4$.

An analysis of the S-box Sb_0 reveals possible values for RC. More precisely, we first find all possible affine invariant subspaces for the S-box[1], that is, $\mathsf{Sb}_0(u \oplus A) = v \oplus A$. Subsequently, if $RC \subseteq A$, then the addition of the round constants lies in A, thus the space is stable. For instance, in the original Midori64, $u = \mathtt{8}, v = \mathtt{8}, A = \{\mathtt{0,1}\}$ and $RC = \{\mathtt{0,1}\} \subseteq A$.

A computer search shows that there are several affine subspaces for Sb_0, some even of size 4 (refer to Table 3). For example, $u = \mathtt{2}$, $v = \mathtt{d}$, $A = \{\mathtt{0,\ 5,\ a,\ f}\}$ is an affine invariant subspace for Sb_0. For this subspace, if $RC \subseteq A$, then the weak-key class would be larger: each subkey cell can take any of the values from A, thus the size of the weak-key class would become 2^{64}.

Table 3: Affine invariant subspaces for Sb_0

u	v	A	u	v	A
0	c	0 c	3	3	0 4
1	a	0 b	4	e	0 a
1	a	0 2 9 b	5	b	0 e
2	d	0 f	5	b	0 2 c e
2	d	0 5 a f	6	f	0 9
3	3	0 b	7	7	0 f
3	3	0 a	7	7	0 e
3	3	0 7 a d	8	8	0 1

The modified constants not only permit distinguishers for larger weak-key classes, but lead to a key recovery for the classes. Note, in our key-recovery attack on Midori64 with the original constants, we have used the fact $\mathsf{Sb}_0(\mathtt{8}) = \mathtt{8}, \mathsf{Sb}_0(\mathtt{9})$, thus it was possible to model the S-box as a simple identity function, which in turn made the whole encryption to behave as a linear function. In general, as long as the S-box behaves as an affine function on the invariant subspace (for the S-box), the key recovery will be reduced to solving a system of linear equations, e.g. any 2-bit permutation is an affine mapping. Let us focus on the above example $u = 2$, $v = d$, $A = \{0, 5, a, f\}$, that is $\mathsf{Sb}_0(\mathtt{2} \oplus \mathtt{0}) = \mathtt{d} \oplus \mathtt{0}, \mathsf{Sb}_0(\mathtt{2} \oplus \mathtt{5}) = \mathtt{d} \oplus \mathtt{a}, \mathsf{Sb}_0(\mathtt{2} \oplus \mathtt{a}) = \mathtt{d} \oplus \mathtt{5}, \mathsf{Sb}_0(\mathtt{2} \oplus \mathtt{f}) = \mathtt{d} \oplus \mathtt{f}$. We need to find a linear function $l(x)$, such that $l(\mathtt{0}) = \mathtt{0}, l(\mathtt{5}) = \mathtt{a}, l(\mathtt{a}) = \mathtt{5}, l(\mathtt{f}) = \mathtt{f}$, hence, $\mathsf{Sb}_0(x) = l(\mathtt{2} \oplus x) \oplus \mathtt{d}$ on the points from A. By solving the system of linear equations $l(\mathtt{5}) = \mathtt{a}, l(\mathtt{a}) = \mathtt{5}$, where l is represented as a binary 4×4 matrix of unknowns, we deduce that $l(x) = l(x_1|x_2|x_3|x_4) = x_2|x_1|x_2|x_1$. Similarly to the previous discussion from Section 3.2, we note that the remaining operations in the cipher are all linear, thus the whole encryption becomes a linear function. Hence, again the key can be recovered by solving a system of linear equations.

The search space can be enlarged to $\mathsf{Sb}_0(u \oplus A) = v \oplus A'$. If $A \cap A'$ does not have intersection other than 0 and $RC \subseteq A \cap A'$ the addition of the round constants lies in A and A', thus the space will be stable. There is a large number of such subspaces when the sizes of A and A' are both two. In addition, we also found two cases where the size of A and A' are four. Those two cases are shown in Table 4.

[1] This is in line with the discussion presented in [LMR15], see Lemma 6.

Table 4: Affine invariant subspaces for Sb_0 with $A \neq A', |A| = |A'| = 4$

u	A	v	A'
c	0 1 2 3	0	0 2 4 6
0	0 5 a f	1	0 7 a d

5 A Search for Strong S-boxes

In this section, we search for S-boxes that resist invariant subspace attacks against a Midori-like structure, which will be detailed later. Our goal is to satisfy simultaneously other general S-box design criteria, in particular, those considered in Midori. Among several criteria, the concept *minimizing depth* deserves carefully attention, and we will explain it in Section 5.1. To resist invariant subspace attacks, criteria for S-boxes depend on what key schedule function is assumed, how strongly attacks are avoided, and on the choice of involution S-boxes/non-involution S-boxes. We first present such classification for case analysis in Section 5.4. Prior to the case analysis, we demonstrate in Section 5.2 that the existence of affine subspaces for an S-box has a very close relation to its differential distribution table (DDT). For example, all subspace transitions for Sb_0 used in Midori64's can be recovered from its DDT. This observation is important to consider as an S-box design criteria. The details for each case analysis are given in Section 5.5 and Section 5.6.

Previous Work on S-box Search

The research on S-boxes has been an ongoing topic for several years. One direction is to analyse and classify the S-boxes based on their cryptographic properties such as resistance against differential and linear attacks. For 4-bit S-boxes, Leander et al. [LP07] have proposed 16 affine equivalence classes of S-boxes that are optimal against differential and linear attacks, so-called *optimal S-boxes*. In 2011, Saarinen [Saa11] has extended the search to all 4-bit S-boxes and has introduced the *golden S-boxes* which are not only optimal against differential and linear attacks, but also have optimal algebraic degree and other properties that may not be preserved under the affine equivalence classes. Another direction is to search for better implementation of existing S-boxes like in [Osv00, Can05, UDCI+11]. In [Osv00, UDCI+11], heuristic searches were conducted to find the minimal sequence of basic operations like XOR/NXOR, AND/OR, NAND/NOR and NOT to implement their target S-boxes. Notable in [Sto16], Stoffelen built the SAT-solver based S-box search tool, which can find a low-depth S-box along with other implementation criteria. In contrast to the previous work, our goal is to search for 4-bit S-boxes which resist not only the classical attacks, but as well the relatively new invariant subspace attacks. In addition, by considering the depth of the S-boxes, we develop an evaluation tool that generates the shortest depth of our target S-boxes (see Section 5.1 for the details). Note that the definition of depth is slightly different between [Sto16] and ours.

5.1 General S-box Design Criteria

Our goal is to find S-boxes that satisfy the following criteria[2]:

- the maximal differential probability is 2^{-2}.

[2]We would like to point out that these (plus the involution property) are the criteria chosen by the designers of Midori. More generally, the discussion and criteria mentioned in this paper should be an additional consideration on top of the other criteria, for instance the algebraic degree, that the designers have when designing ciphers, especially for lightweight ciphers like Midori.

- the maximal absolute bias of a linear approximation is 2^{-2}.

When there are several candidates, we pick the S-box that has:

- the smallest number of fixed points,
- the smallest depth.

Let us clearly define the notion of depth of an S-box. The designers of Midori introduce the metric *depth* to estimate the path delay of S-boxes as:

Definition 1 (Depth, [BBI+15])**.** The depth is defined as the sum of sequential path delays of basic operations AND, OR, XOR, NAND, NOR, XNOR and NOT.[3]

To maintain consistency, we follow the same assumptions of depth and gate size (GEs) for each basic operation as that in [BBI+15]. The depths as well as the required gates of XOR/XNOR, AND/OR, NAND/NOR and NOT are weighted as 2, 1.5, 1 and 0.5, respectively. For example, the depth of the following function is 3.5.

$$((c \text{ NAND } d) \text{ NAND } (b \text{ NAND } (\text{ NOT } a))) \text{ NOR } (b \text{ NOR } (a \text{ NAND } d))$$

5.1.1 S-box Depth Evaluation Tool

The designers of Midori discuss only how to evaluate the depth of an S-box that has already been described by the aformentioned logical operations. However, the details on finding an S-box description with small (or smaller) depth are omitted. This motivates us to develop an *S-box depth evaluation tool*, which takes as input an S-box (given in tabular form) and outputs its representation of the logical operations that has a certain depth[4] (or detects that no such representation exist). The tool has helped us to find S-boxes presented in the latter part of this section. We provide our tool as an auxiliary supplemental material to this submission, and will open it to the public as it will help future designers to identify good S-boxes. In the following, we explain the concept behind it.

The tool evaluates the depth of a given S-box by matching against values from precomputed look-up tables. The tables are created as follows. We generate all Boolean-function representations of each output bit of an S-box that has a certain depth, and store them in a table. Hence, all representations within the same table have the same depth.

We use a recursive algorithm to enumerate all Boolean functions of four input bits that can be expressed with certain depth. For each Boolean function, we record four types of information;

- truth table for all 4-bit inputs $0000, 0001, 0010, \ldots, 1111$,
- expression,
- depth,
- gate size.

For instance, 1010101010101010 represents the truth table of a Boolean function "NOT d" where $a|b|c|d$ is the 4-bit input to the S-box, and the depth and gate size are both 0.5, which is recorded as [1010101010101010, "NOT d", 0.5, 0.5].

The recursive algorithm to enumerate all Boolean functions works as follows:

[3]The original definition of *depth* in [BBI+15] does not contain XOR and XNOR, but XOR appears in its example and XNOR is mentioned in gate estimations. We consider both XOR and XNOR here and assume the depth of XNOR is 2.

[4]Recently, Biryukov and Perrin [BP15] have developed a very powerful tool for analysis of S-boxes, but their motivation differs from ours.

Depth 0: the four input bits themselves are four Boolean functions.

Depth 0.5: apply basic operation NOT to the 0-depth Boolean functions.

Depth 1: apply basic operation NAND/NOR to two different 0-depth Boolean functions.

Depth 1.5: there are three ways to generate Boolean functions: 1) apply basic operation AND/OR to two different 0-depth Boolean functions, 2) apply basic operation NAND/NOR to two different 0.5-depth Boolean functions, and 3) apply basic operation NOT to two different 1-depth Boolean functions.

Depth $d \geq 2$: there are four ways to generate required Boolean functions: apply basic operations XOR/XNOR, AND/OR, NAND/NOR and NOT to Boolean functions with depth $d-2, d-1.5, d-1, d-0.5$ respectively.

For some Boolean functions, there are several expressions with the same depth. In this case, we store the one with the smallest gate size. For expressions representing the same Boolean function with the same depth and the same gate size, we treat them as equivalent and store only the first.

Each output bit of a 4-bit S-box is a balanced Boolean function of the four input bits. Therefore, we filter out the unbalanced, and store only the balanced to test for S-boxes. Table 5 gives the number of balanced Boolean functions that can be expressed with a small depth. The number of all 4-bit balanced Boolean functions is $12870 \approx 2^{13.65}$, while our enumeration for depth of 4.5, which also includes depth of less than 4.5, enumerates $12806 \approx 2^{13.64}$ functions. Thus, most of the 4-bit balanced Boolean functions can be expressed with depth 4.5.

Table 5: Number of balanced Boolean functions with respect to expression depth

Depth	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5
Number	2^2	2^2	0	0	2^4	$2^{6.86}$	$2^{8.39}$	$2^{10.72}$	$2^{13.43}$	$2^{13.64}$

Given an S-box, by calculating the truth table of each output bit, and subsequently matching against the created look-up tables, we find its expressions with least depth and gate size. The depth of the given S-box is estimated as the largest depth of the output bits. We use the stored balanced Boolean functions to construct S-boxes of certain depth in two steps:

Step 1. Search for pairs of balanced Boolean functions with certain depth that satisfy 2-bit balance, i.e., the weights of $00, 01, 10, 11$ are equal in truth table. For instance,

$$\begin{pmatrix} 1100110011001100 \\ 1111111100000000 \end{pmatrix}$$

is a pair of 2-bit balanced Boolean functions.

Step 2. Search for combinations of 2-bit balanced pairs of Boolean functions that make a permutation, i.e. an S-box, and simultaneously satisfy that the maximal probability of differential is 2^{-2} and the maximal absolute bias of a linear approximation is 2^{-2}.

With the tool, we have generated all $2^{25.2}$ S-boxes with depth of at most 3.5, among which $2^{11.17}$ are of depth 3.

5.2 The Relation between DDT and Invariant Subspace of an S-box

Affine invariant subspaces for an S-box are closely related to its differential distribution table (DDT). The existence of the affine subspace transitions $u_1 \oplus A \xrightarrow{S} u_2 \oplus A'$ with low dimension of A and A' immediately provides information about DDT and vice versa. In particular, for a 4-bit S-box with maximal differential probability of 2^{-2}, all subspace transitions can be recovered only from DDT.

We show in this section the relation between affine subspace transitions and DDT of an S-box, and explain how to use DDT to search for S-boxes that can be used to resist invariant subspace attacks for the whole cipher.

5.2.1 Deriving DDT from Low Dimension Affine Subspace

If A is a vector space of dimension 2 or less, the number of elements in A will appear in DDT. Suppose that there exists an affine subspace transition $u_1 \oplus A \xrightarrow{S} u_2 \oplus A'$. It means that for any input $x \in A$, the S-box can be seen as $S(u_1 \oplus x) = l(u_1 \oplus x) \oplus u_2 = l(x) \oplus u$, where $l(x)$ is a linear function that transforms A into A' and u is a constant calculated as $u = l(u_1) \oplus u_2$. As $l(x)$ is a linear function, for any input difference it has a differential probability of one.

For instance, let us consider the vector space with dimension 1, i.e. $u_1 \oplus A \xrightarrow{S} u_2 \oplus A'$, where $A = \{0, v\}$ and $A = \{0, v'\}$. This is simply converted into $v \xrightarrow{S} v'$. When the difference of two values in the vector space is considered, it suggests $\Delta_{in} \xrightarrow{S} \Delta_{out}$ where $\Delta_{in} = v$ and $\Delta_{out} = v'$. In the end, we have 2 for the entry $(\Delta_{in}, \Delta_{out}) = (v, v')$ in DDT.

The same is applied to a vector space with dimension 2, $A = \{0, v_1, v_2, v_1 \oplus v_2\}$ and $A' = \{0, v'_1, v'_2, v'_1 \oplus v'_2\}$. Differently from dimension 1, there are three ways to make the input difference which are $\Delta_1 = 0 \oplus v_1$, $\Delta_2 = 0 \oplus v_2$, and $\Delta_3 = 0 \oplus v_1 \oplus v_2$ (all result in different output difference). Thus we will have 3 different entries with element 4. By setting $\Delta'_1 = S(0) \oplus S(v_1)$, $\Delta'_2 = S(0) \oplus S(v_2)$, and $\Delta'_3 = S(0) \oplus S(v_1 \oplus v_2)$, the entries for (Δ_1, Δ'_1), (Δ_2, Δ'_2) and (Δ_3, Δ'_3) will be 4.

5.2.2 Deriving Affine Subspace from DDT

Affine subspaces can be derived from DDT up to vector space level. Having 2 for the entry of $(\Delta_{in}, \Delta_{out}) = (v, v')$ in DDT indicates that there are exactly 2 input values such that $S(x) \oplus S(x \oplus v) = v'$. By setting $u_1 \leftarrow x$ and $u_2 \leftarrow S(x)$, it can be described as $u_1 \oplus \{0, v\} \xrightarrow{S} u_2 \oplus \{0, v'\}$. Without the exact specification of the S-box, DDT does not provide the value of x and $S(x)$ satisfying the difference transition. Therefore, the offset values u_1 and u_2 cannot be recovered only from DDT.

The case of the value of 4 in DDT is basically the same. Having 4 for the entry of $(\Delta_{in}, \Delta_{out}) = (v, v')$ in DDT indicates that there are exactly 4 input values such that $S(x) \oplus S(x \oplus v) = v'$, i.e. $x \in \{x_1, x_1 \oplus v, x_2, x_2 \oplus v\}$. Let u_1 be x_1 and w be $x_1 \oplus x_2$. Then, 4 input values can be described as $u_1 \oplus \{0, v, w, w \oplus v\}$. Similarly, let u_2 be $S(x_1)$ and w' be $S(x_1) \oplus S(x_2)$. Then, 4 output values can be described as $u_2 \oplus \{0, v', w', w' \oplus v'\}$. Consequently, the affine subspace $u_1 \oplus \{0, v, w, w \oplus v\} \xrightarrow{S} u_2 \oplus \{0, v', w', w' \oplus v'\}$ holds. Note that three different entries with value 4 lead to an identical affine subspace of dimension 2. Namely, not only $(\Delta_{in}, \Delta_{out}) = (v, v')$ but also $(\Delta_{in}, \Delta_{out}) = (w, w')$ and $(v \oplus w, v' \oplus w')$ lead to $u_1 \oplus \{0, v, w, w \oplus v\} \xrightarrow{S} u_2 \oplus \{0, v', w', w' \oplus v'\}$.

In Appendix A, we demonstrate how to recover all the affine subspaces of Sb_0 used in Midori64 from its DDT.

5.2.3 Remarks on Affine Subspace with Higher Dimension

The above discussion does not apply to affine subspace transitions with dimension of 3. This is because the 8 input values that correspond to 4 pairs that share the same output difference through the S-box do not necessarily form an affine space. to be more precise, suppose that DDT has value 4 for the entry of $(\Delta_{in}, \Delta_{out}) = (v, v')$, indicating that there exist four input values x_0, x_1, x_2, x_3 such that

$$\begin{aligned} S(x_0) \oplus S(x_0 \oplus v) &= v', \\ S(x_1) \oplus S(x_1 \oplus v) &= v', \\ S(x_2) \oplus S(x_2 \oplus v) &= v', \\ S(x_3) \oplus S(x_3 \oplus v) &= v'. \end{aligned}$$

Let u_1 be x_0, u_2 be $S(x_0)$, w be $x_0 \oplus x_1$, w' be $S(x_0) \oplus S(x_1)$, t be $x_0 \oplus x_2$, and t' be $S(x_0) \oplus S(x_2)$. The necessary and sufficient condition that the above forms an affine space of dimension 3 (bases and offset are (v, w, t) and u_1 for input and (v', w', t') and u_2 for output) is $x_0 \oplus x_3 = v \oplus w$ and $S(x_0) \oplus S(x_3) = v' \oplus w'$. This is not always true. This fact yields a few remarks that deserve attention.

The absence of affine subspace transition with dimension 3 for 4-bit S-boxes does not imply that the maximal differential probability is 2^{-2}, i.e. does not ensure that all elements in DDT are at most 4. This is because an affine subspace transition with dimension 1 and one with dimension 2 can impact to an identical entry in DDT, thus the number in this entry becomes 6.

Similarly, ensuring the maximal differential probability of 2^{-2} for a 4-bit S-box, in other words, having 4 or less in all entries of DDT, does not imply that there is no affine subspace transition with dimension 3 (or higher for larger S-boxes). For example, a transition with dimension 3 can be composed of two transitions with dimension 2.

In short, we have proven the following proposition.

Proposition 2. *For an n-bit S-box with maximal differential probability 2^{-n+2}, every affine subspace transition with dimension 2 corresponds to three entries of 4 in DDT.*

5.2.4 Early Detection of Higher Dimension Affine Subspaces from DDT

As discussed in the earlier section, there is no clear relation between DDT and the affine subspace transitions with dimension higher than 2. Nonetheless, by observing DDT of an S-box we can still have some form of an early detection of higher dimension affine subspaces. Suppose that the highest affine subspace transition is of dimension 2, then by Proposition 2, the number of entries of 4 in DDT will be a multiple of 3.

Corollary 1. *For an n-bit S-box with maximal differential probability 2^{-n+2}, if the number of entries of 4 in DDT is not a multiple of 3, then there exists an affine subspace transition with dimension higher than 2.*

5.3 Target Structure

In the following discussion, we mainly discuss the SPN-type cipher whose round function consists of the following operations.

1. A subkey is XORed to the whole state. We consider three types of key schedule that will be explained later.

2. A 4-bit S-box is applied to the entire block in parallel. We consider two types of S-boxes, involution and non-involution. Some of the discussions can be extended to an S-box of any size. We will mention it explicitly for this case.

3. A linear layer may be applied. To discuss the S-box criteria that stand independently of the choice of the linear layer, we assume the worst case, i.e. affine space of the input to the linear layer does not change through the linear layer. This stands against an identity map and the linear layer of Midori64.

5.4 Classification for Case Analysis

Invariant subspace attacks on Midori64 exploit the property that if all the cells of the state are in the same affine subspace, then the linear layer MixColumn $\circ$ ShuffleCell preserves this subspace. Thus, the resistance against invariant subspace attacks should be evaluated by considering the S-box, the key schedule function and the round constants.

In [LMR15], Leander et al. point out that the choice of proper round constants prevents invariant subspace attacks (or makes them probabilistic, thus by increasing the number of rounds, they can be avoided). Hence, altering round constants in Midori64 is the first, and perhaps the easiest, choice to stop our attacks. That is, instead of the current values of 0 and 1 for the cells of the round constants, one can assign random values for the cells (or even random values of Hamming weight not exceeding one), and expect resistance against invariant subspace attacks after certain number of rounds.

It is possible to turn the problem upside down, and examine the case when the constants are worse (with respect to invariant subspace attacks), but still we expect some level of protection against this type of attacks. We have seen from Section 4 that the altered Midori64 round constants lead to larger weak-key classes. In addition, altered key schedules may lead to even larger classes. Can we make sure that regardless of the round constants and of the key schedule (to a certain extent), a proper choice of an S-box may stop invariant subspace attacks or may lead to such an attack but on only a small subset of keys? This line of research brings us a step closer to a *provable security against invariant subspace attacks*: it suffices to examine only the S-box (or to choose a good S-box), in order to stop the attacks or to limit their applicability to a small set of weak keys. We will examine the security in respect to the invariant subspace attacks as the one presented on Midori64.

In the remaining of the section, we examine the classes of S-boxes that ensure resistance against invariant subspace attacks. We split the analysis according to three criteria as presented further.

5.4.1 Choice of Involution/Non-involution S-box

In general, involution S-boxes have a lower implementation cost for the whole cipher compared to non-involution S-boxes (do not require implementation of inverse), while non-involution S-boxes have higher security. We will show that this principle also stands with respect to resistance against invariant subspace attacks.

5.4.2 Classes of Key Schedule Function

In general, invariant subspace attacks can be prevented by using a strong key schedule function. On the other hand, practical designs of lightweight cryptography use a light key schedule function as in Midori. We consider the following three classes of key schedule functions:

KSF1: A single key K is used in every round, e.g. Midori128 and LED-64 [GPPR11].

KSF2: Two keys K_1 and K_2 are alternately used in every two rounds, e.g. Midori64 and LED-128.

KSF3: No assumption on the key schedule function.

We will see that KSF1 is relatively easy to protect, i.e. finding suitable S-boxes is easy, because of the limited degrees of freedom that the attacker is given (can choose only weak keys of K). In KSF2 the attacker can choose weak keys on K_1 and K_2 independently, thus protecting KSF2 is harder than KSF1. Finally, even though KSF3 is extremely hard to protect, we still can find S-boxes resisting strong invariant subspace attacks.

5.4.3 Degree of Resistance

S-box design criteria also depend on how "strongly" designers want to avoid attacks. As described in previous sections, invariant subspace attacks usually work only for a fraction of the entire key space, or weak keys. To ensure that the number of weak keys is only one is harder than to ensure that the number of weak keys is limited to a small size.

Goal1: The goal is to ensure that the number of weak keys is limited to $c \cdot 2^b$, where c is a small constant, and b is the number of cells in the key (e.g. for Midori64 this is roughly 2^{32}).

Goal2: The goal is to ensure that the number of weak keys is only one.

If some S-box achieves certain degree of resistance under KFS2, it can also achieve that degree of resistance, or even better, under KFS1. On the other hand, if an S-box could not achieve a particular degree of resistance under KFS2, neither will it under KFS3. Similarly, we can see that Goal2 is of a higher degree resistance against invariant subspace attack than Goal1.

5.5 A Search for Strong Involution S-boxes

Further, we examine the cases of involution S-boxes that provide a certain degree of resistance (Goal1 and Goal2) against invariant subspace attack under the aforementioned classes of key schedule function.

5.5.1 Impossibility for KFS2 and KFS3

An involution S-box, irrespectively of its size, cannot achieve Goal1 (and thus Goal2) under KSF2 (and thus under KSF3). That is, for any choice of an involution S-box, it is impossible to prove that the number of weak-keys in an invariant subspace attack is limited to only $c \cdot 2^b$ (which is Goal1) in a cipher that has alternating subkeys (which is KSF2), without considering the round constants.

The main reason lies in the fact that any affine subspace transition is both ways due to its involution property of the S-box, i.e. $u_1 \oplus A_1 \overset{S}{\longleftrightarrow} u_2 \oplus A_2$. DDT of the S-box must contain entries of 2, thus there exist A_1 and A_2 with dimension of 1. (In case of 4-bit S-boxes, as mentioned earlier in Section 5.2, DDT must contain entries of 4, thus there exist A_1 and A_2 with dimension of 2.) Let $K_1 \in A_2$ and $K_2 \in A_1$ – there are 2^b such keys. Then an attacker can launch a 2-iteration invariant subspace attack if the plaintext belongs to the affine subspace $u_1 \oplus A_1$. More specifically, after the first substitution layer, the affine subspace $u_2 \oplus A_2$ is XOR-ed with K_1 which does not change the affine subspace. The affine subspace is then transformed back to $u_1 \oplus A_1$ under the second substitution layer, which again remains unchanged after adding K_2, and this cycle repeats. Hence, the ciphertext will belong to $u_1 \oplus A_1$ or to $u_2 \oplus A_2$, depending on the parity of the number of rounds, and thus the attack is possible under 2^b weak keys (2^{2b} weak keys for 4-bit S-boxes).

5.5.2 Impossibility for KFS1 with Goal2

Involution S-boxes always allow weak-key classes of size 2^b under KSF1, hence no involution S-box can achieve Goal2 under KSF1. The property stands irrespectively of the size of the S-box.

Consider an element x such that $S(x) \neq x$. Note, such element always exists unless the S-box is simply an identity mapping. Since $S(S(x)) = x$, affine subspace transition $u \oplus A \xleftrightarrow{S} u \oplus A$ of dimension 1 always exists, i.e. $u \oplus \{0, x \oplus S(x)\} \xleftrightarrow{S} u \oplus \{0, x \oplus S(x)\}$, where $u \in \{x, S(x)\}$. Thus a subkey K with cells from A will be weak with respect to invariant subspace attack. The number of such keys is at least 2^b, which contradicts with Goal2.

5.5.3 KSF1, Goal1

With a proper choice of an S-box, we can achieve Goal1 with KSF1, i.e. we can show that the number of weak keys for invariant subspace attack is limited to only 2^b if the cipher has identical subkeys, without considering the round constants (that is, for any round constants).

As discussed in Section 5.2, if DDT of an S-box has nonzero entry for a pair of input and output differences (v, v'), then there exists an affine subspace transition $u_1 \oplus A_1 \xrightarrow{S} u_2 \oplus A_2$, where $A_1 = \{0, v\}$ and $A_2 = \{0, v'\}$. However, under KSF1, an invariant subspace holds if and only if $A_1 = A_2$ and $K = u_1 \oplus u_2 \oplus A_1$. Therefore, we can achieve Goal1 under KSF1 if we can avoid nonzero entries in the diagonal of DDT of an S-box.

From the earlier section we have seen that involution S-box always permits an affine subspace of dimension 1, however, we can avoid affine subspaces with dimension 2 by searching for involution S-boxes with no entries of 4 on the diagonal of DDT. According to the criteria from Section 5.1, we search for candidate S-boxes with minimal number of fixed points and minimal depth. We found a few S-boxes with only 2 fixed points and a depth of 4 (cf. Sb_0 in Midori64 with 4 fixed points and depth of 3.5):

S_1^{new} : `1 0 4 6 2 8 3 9 5 7 d b e a c f` (fixed points: `b,f`)
S_2^{new} : `1 0 4 6 2 8 3 9 5 7 a f e d c b` (fixed points: `a,d`)
S_3^{new} : `1 0 4 3 2 8 6 9 5 7 d f e a c b` (fixed points: `3,6`)

As a proof-of-concept, we list the Boolean-function representation of the first S-box ($a|b|c|d$ is the input and $a'|b'|c'|d'$ is the output), which clearly shows that the depth is 4:

$$
\begin{aligned}
a' &= (a \text{ NAND } (b \text{ OR } c)) \text{ NAND } (b \text{ NAND } d),\\
b' &= (a \text{ NOR } (b \text{ NOR } (\text{NOT } c))) \text{ NOR } ((a \text{ NAND } d) \text{ NOR } (b \text{ XNOR } c)),\\
c' &= (b \text{ NAND } (a \text{ XNOR } d)) \text{ NAND}\\
&\quad ((b \text{ NAND } c) \text{ NAND } ((a \text{ NOR } c) \text{ NOR } (b \text{ NOR } d))),\\
d' &= ((c \text{ NOR } d) \text{ NOR } (a \text{ OR } b)) \text{ NOR}\\
&\quad ((a \text{ NOR } (\text{NOT } c)) \text{ NOR } (b \text{ NAND } (c \text{ NAND } d))).
\end{aligned}
$$

Note, the depth (resp. the GEs) for the outputs a',b',c',d' are $3.5, 4, 4, 4$ (resp. $4.5, 7.5, 9, 9$).

5.6 A Search for Strong Non-involution S-boxes

As we have seen, non-zero entries on the diagonal of DDT of involution S-boxes, correspond to iterative invariant subspaces of certain dimension, and it is impossible to avoid them. Hence, it is meaningful to consider non-involution S-boxes. Below, we show that for this type of S-boxes achieving Goal2 is indeed possible, under some key schedules. Furthermore, we give the best achievable goals under each key schedule function.

5.6.1 KSF3 and KSF2, Goal1

To achieve Goal1, two conditions are imposed on the S-box:

1. There are no affine subspace transitions of dimension more than 2.
2. There are no affine subspace transitions of dimension 2 that can be connected (output subspace of one coincides with input subspace of another).

S-boxes satisfying the above 2 conditions will allow only invariant subspaces of dimension at most 1, i.e., they achieve Goal1. Condition 2 is particularly important as it assures that one cannot build iterative affine subspace characteristic. For 4-bit S-boxes, the only proper affine subspaces of $\{0,1\}^4$ with dimension greater than 2 are those of dimension 3, and there are 15 such subspaces in total, hence an exhaustive verification can be done instantaneously. To fulfill Condition 2, we list all affine subspace transitions $u_1 \oplus A_1^i \xrightarrow{S} u_2 \oplus A_2^i$ with A_1^i and A_2^i of dimension 2, for $i = 1, 2, \ldots, l$. If $A_1^i \neq A_2^j$ for all $i, j = 0, 1, \ldots, l$, i.e., there is no common input and output affine subspaces, then A_2^i can not be mapped to any dimension 2 affine subspace in the next round. Computer search shows such S-boxes do exist, and there are many of them. We start with the 16 *optimal S-boxes* introduced in [LP07]. Among other criteria, these S-boxes have the best possible differential/linear properties. We find there are five of them fulfilling the above two conditions:

```
0 1 2 d 4 7 f 6 8 e b 5 a 9 3 c
0 1 2 d 4 7 f 6 8 e c 9 5 b a 3
0 1 2 d 4 7 f 6 8 e 9 5 a b 3 c
0 1 2 d 4 7 f 6 8 c 5 3 a e b 9
0 1 2 d 4 7 f 6 8 b e 3 a c 5 9
```

As mentioned before, Saarinen found golden S-boxes out of the 16 optimal S-boxes [Saa11]. Interestingly, all of the above 5 S-boxes do not have any intersection with Sarrinen's golden S-boxes. For the purpose of giving convenience to future research, we call the above 5 S-boxes *silver S-boxes*.

Notice that XORing some value to the input/output of the S-boxes only changes the offset values of the affine subspaces while the subspaces remain unchanged, thus still satisfies the above conditions. Simply considering XORing some value to the input/output of the S-boxes, there are a total of 528 S-boxes with no fixed point. Here, we did not enlarge the search space to $P \circ S \circ Q$ for linear transformations P and Q. This is because such P and Q require to change MixColumn and then discussion exceeds the choice of the S-box. As a result, we checked the depth of 528 S-boxes, and found that none of them achieves depth 3.5. Hence, minimum depth is 4 and the following S-box is one of them.

$$S_4^{\text{new}} : \texttt{5 4 7 8 1 2 a 3 d b e 0 f c 6 9}$$

In Appendix B, we list Boolean functions to compute each output bit of S_4^{new} with depth 4.

5.6.2 Impossibility for KSF3 and KSF2 with Goal2

To achieve Goal2 under KSF2 and KSF3, two types of transitions should be avoided, i.e., $\Delta \xleftrightarrow{S} \Delta'$ with first type of differences $\Delta \neq \Delta'$ and second type of differences $\Delta = \Delta'$. While the first type transition corresponds to symmetric non-zero entries with respect to the diagonal of DDT, and could form 2-transition invariant subspace of the form $\Delta \xrightarrow{S} \Delta' \xrightarrow{S} \Delta$, the second type transition corresponds to non-zero entries on the

diagonal and 1-transition invariant subspace $\Delta \xrightarrow{S} \Delta$ with $\Delta \neq 0$. We find none of the 16 optimal S-boxes avoids these two transitions. Furthermore, we extend each of the 16 optimal S-boxes S by prepending and appending an invertible linear layer P and Q to the input and output of the S-box, respectively. The extended S-boxes are of the form $P \circ S \circ Q$ so that new S-boxes inherit the good differential and linear properties from S itself. However, none of the extended S-boxes avoids both types of transitions in DDT simultaneously.

5.6.3 KSF1 with Goal2

Note that Goal1 under KSF1 can be achieved with the same approach as Goal1 with KSF2 and KSF3. To achieve Goal2 under KSF1, there exists a simple method: we need to ensure that all dimension 1 invariant subspaces for the S-box cannot be used in the invariant subspace attack on the cipher. Since only 1-iteration transitions are useful here, i.e., those entries in the diagonal of DDT with input and output differences of the S-box being the same, we search for S-boxes with no non-zero entries in the diagonal. The search space is of the form $P \circ S \circ Q$, where S is one of the 16 optimal S-boxes. Note, there are roughly 2^{19} candidates S-boxes that achieve Goal1. Furthermore, we add constants to the input and output of the S-boxes to reduced the number of fixed points. We list below an S-box found by program search that has a single fixed point and a depth of 4, generated from the first optimal S-box. The detailed Boolean functions to compute each output bit is postponed to Appendix B.

$$S_5^{\text{new}} : \texttt{0 d 6 5 7 b e a 2 8 3 c 9 f 1 4}$$

5.7 Discussion on the S-box Search Results

The results of the search for S-boxes are summarized in Table 6.

Table 6: Existence of S-boxes that prevents invariant subspace attacks

		Involution	Non-involution
KSF1	**Goal1**	✓	✓
	Goal2	-	✓
KSF2	**Goal1**	-	✓
	Goal2	-	-
KSF3	**Goal1**	-	✓
	Goal2	-	-

5.7.1 Involution S-boxes and Evaluation of Sb_0

Table 6 clearly illustrates that resisting invariant subspace attacks only by choosing a proper involution S-box is very hard due to the involution property. Secure constructions based on S-box analysis exist only for Goal1 with KSF1. In Table 7 we compare one such S-box, e.g. S_1^{new}, and the original S-box Sb_0 used in Midori64.

Table 7: Comparison of involution S-boxes

	Depth	Goal1	Optimal security	#fixed points	Gate size
Midori64	3.5	-	✓	4	23
Ours	4	✓	✓	2	30

The new S_1^{new} ensures that the maximal size of weak keys against invariant subspace attacks is upper bounded by around 2^b independently of the choice of round constants. In comparison, Sb_0 allows several weak-key classes of size 2^{2b} under a bad constant choice. Our S-box as well has a smaller number of fixed points, and slightly larger depth (the exhaustive search presented in Section 5.1 reveals that no depth 3.5 involution S-box exist that satisfies all of our requirements). Note, the column of "Optimal security" means that the S-box achieves the best possible security among all of the S-boxes with the same depth.

The S-boxes we have found require relatively large gate size. For example, the S-box Class13 identified in [UDCI+11] requires only 10.5 GEs. Searching for S-boxes that are efficient in both of depth and gate size is an open problem. Meanwhile, there should be many environments that energy is crucial while extra hundreds of GEs are acceptable. Our S-boxes are very suitable in such a situation.

5.7.2 Non-Involution S-boxes

In contrast to involution S-boxes, there exist non-involution S-boxes that provide resistance against invariant subspace attacks under KSF1, e.g., S_5^{new}. Under KSF2 and even KSF3 (an arbitrary key schedule), they still provide resistance up to a small set of weak keys – refer to the S-box S_4^{new}. The minimum depth of the S-boxes in both cases is 4, and the number of fixed points is as low as one.

6 Concluding Remarks

We have presented an invariant subspace attack against full Midori64. We have shown that Midori64 has a class of 2^{32} weak keys, and with such keys along with a properly chosen plaintext, the cipher becomes a linear transformation thus can be distinguished with a single chosen-plaintext query. Furthermore, the key recovery can be performed simply by solving a system of linear equations.

We have also discussed the topic of provable security against invariant subspace attacks against a certain type of the SPN structure. In certain scenarios, the resistance can be achieved only by focusing on the S-boxes, regardless of the choice of the round constants. With respect to 4-bit S-boxes, several non-involution ones help to achieve sufficiently high level of security against such attacks.

At the current stage, the attack cannot be applied to Midori128. The difficulty comes from the usage of four different S-boxes, SSb_0, SSb_1, SSb_2 and SSb_3. To apply the invariant subspace attack, all the S-boxes must have an identical affine subspace transition, and this is unlikely to occur.

Acknowledgments

We would like to thank anonymous reviewers for their fruitful comments. Siang Meng Sim is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BP15] Alex Biryukov and Léo Perrin. On reverse-engineering S-boxes with hidden design criteria or structure. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 116–140. Springer, 2015.

[Can05] David Canright. A very compact S-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2011.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of Robin, iSCREAM and Zorro. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 254–283. Springer, 2015.

[LP07] Gregor Leander and Axel Poschmann. On the Classification of 4 Bit S-Boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[LW15] Li Lin and Wenling Wu. Meet-in-the-Middle Attacks on Reduced-Round Midori-64. Cryptology ePrint Archive, Report 2015/1165, 2015.

[Osv00] Dag Arne Osvik. Speeding up Serpent. In *AES Candidate Conference*, pages 317–329, 2000.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4 x 4-bit S-boxes. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2011.

[Sto16] Ko Stoffelen. Optimizing s-box implementations for several criteria using SAT solvers. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 140–160. Springer, 2016.

[UDCI+11] Markus Ullrich, Christophe De Canniere, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding optimal bitsliced implementations of 4 x 4-bit S-boxes. In *SKEW 2011 Symmetric Key Encryption Workshop, Copenhagen, Denmark*, pages 16–17, 2011.

A Recovering All Affine Subspace Transitions from DDT

In this section, we demonstrate how to recover all the affine subspace transitions of Sb_0 used in Midori64 (up to vector space level) only from DDT. First of all, DDT of Sb_0 is given in Table 8. Note that Sb_0 is an involution, thus DDT is symmetric. Namely, for any entry of DDT, another entry in the transposed position always has the same value. Also note that Sb_0 was generated to satisfy the maximal differential probability, thus numbers in any entry are less than or equal to 4.

As discussed in Section 5.2, any entry $(\Delta_{in}, \Delta_{out}) = (v, v')$ corresponds to a dimension 1 affine subspace transition $u_1 \oplus \{0, v\} \xrightarrow{S} u_2 \oplus \{0, v'\}$. Hence, it is very easy to recover all affine subspace transitions with dimension 1. For example, the affine subspace transition that we used in the attack, $\mathtt{8} \oplus \{\mathtt{0}, \mathtt{1}\} \xrightarrow{S} \mathtt{8} \oplus \{\mathtt{0}, \mathtt{1}\}$ corresponds to element 2 for $(\Delta_{in}, \Delta_{out}) = (\mathtt{1}, \mathtt{1})$. In other words, only by looking the entry of $(\Delta_{in}, \Delta_{out}) = (\mathtt{1}, \mathtt{1})$ in DDT, we can conclude that no other invariant subspace which is consistent with round constant of Midori64 exists.

Recovering transitions with dimension 2 is more difficult because one transition with dimension 2 corresponds to 3 different entries of DDT with element 4. Hence, we need to

Table 8: Differential Distribution Table (DDT) of Sb_0 used in Midori64. Superscript alphabets show groups of entries that correspond to an identical subspace transition with dimension 2.

Δ_{out} \ Δ_{in}	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	4^E	0	2	2	2	0	2	0	0	0	0	0	2	0
2	0	4^E	0	0	4^E	0	0	0	0	4^C	0	0	4^B	0	0	0
3	0	0	0	0	2	0	4^E	2	2	2	0	0	0	2	0	2
4	0	2	4^E	2	2	2	0	0	2	0	0	2	0	0	0	0
5	0	2	0	0	2	0	0	4^F	0	2	4^A	0	2	0	0	0
6	0	2	0	4^E	0	0	0	2	2	0	0	0	2	2	0	2
7	0	0	0	2	0	4^F	2	0	0	0	0	2	0	4^D	2	0
8	0	2	0	2	2	0	2	0	0	2	0	2	2	0	2	0
9	0	0	4^C	2	0	2	0	0	2	2	0	2	2	0	0	0
a	0	0	0	0	0	4^A	0	0	0	0	4^D	0	0	4^F	0	4^F
b	0	0	0	0	2	0	0	2	2	2	0	4^C	0	2	0	2
c	0	0	4^B	0	0	2	2	0	2	2	0	0	2	0	2	0
d	0	0	0	2	0	0	2	4^D	0	0	4^F	2	0	0	2	0
e	0	2	0	0	0	0	0	2	2	0	0	0	2	2	4^B	2
f	0	0	0	2	0	0	2	0	0	0	4^F	2	0	0	2	4^A

detect which of 3 different entries imply an identical affine subspace transition. We start from finding triplets related to the diagonal of DDT.

Firstly, we focus on the entry $(\Delta_{in}, \Delta_{out}) = (\mathtt{f}, \mathtt{f})$. Considering that Sb_0 is an involution, this indicates that there exists a transition of the following form.

$$u_1 + \{0, \Delta_1, \Delta_2, f\} \xleftrightarrow{S} u_1 + \{0, \Delta_1, \Delta_2, f\}, \qquad \Delta_1 \oplus \Delta_2 = f.$$

Therefore, DDT must have elements 4 in the entry of $(\Delta_{in}, \Delta_{out}) = (\Delta_1, \Delta_2)$ in which $\Delta_1 \oplus \Delta_2 = f$. Then, there is only once case $(\Delta_1, \Delta_2) = (\mathtt{5}, \mathtt{a})$. In Table 8, those triplets are denoted by superscript A. This affine subspace indeed corresponds to $A = \{\mathtt{0}, \mathtt{5}, \mathtt{a}, \mathtt{f}\}$ in Table 3.

With the same analysis, $(\Delta_{in}, \Delta_{out}) = (\mathtt{e}, \mathtt{e})$ leads to $A = \{\mathtt{0}, \mathtt{2}, \mathtt{c}, \mathtt{e}\}$ in Table 3, $(\Delta_{in}, \Delta_{out}) = (\mathtt{b}, \mathtt{b})$ leads to $A = \{\mathtt{0}, \mathtt{2}, \mathtt{9}, \mathtt{b}\}$, and $(\Delta_{in}, \Delta_{out}) = (\mathtt{a}, \mathtt{a})$ leads to $A = \{\mathtt{0}, \mathtt{7}, \mathtt{a}, \mathtt{d}\}$.

The remaining is no longer invariant (iterative with cycle 1) because they are not in diagonal. As long as we have $u_1 + \{0, a, b, a \oplus b\} \xrightarrow{S} u_2 + \{0, x, y, x \oplus y\}$, we immediately obtain another subspace $u_2 + \{0, x, y, x \oplus y\} \xrightarrow{S} u_1 + \{0, a, b, a \oplus b\}$ due to the involution property. Thus, it is natural to consider 6 entries with element 4 in one group.

We then focus on $(\Delta_{in}, \Delta_{out}) = (\mathtt{1}, \mathtt{2})$ and its inverse $(\mathtt{2}, \mathtt{1})$. Because there is no clue that which of differential transitions belong to the same group, we do the exhaustive test. When we pick (x, y) (and thus (y, x) for inverse), both of $(\mathtt{1} \oplus x, \mathtt{2} \oplus y)$ and $(\mathtt{2} \oplus y, \mathtt{1} \oplus x)$ must have 4 in DDT. For example, we pick $(\mathtt{5}, \mathtt{7})$ (and thus $(\mathtt{7}, \mathtt{5})$ for inverse). Then, $(\mathtt{1} \oplus \mathtt{5}, \mathtt{2} \oplus \mathtt{7}) = (\mathtt{4}, \mathtt{5})$ does not have elements 4 in DDT, showing that $(\mathtt{5}, \mathtt{7})$ is not in the same group as $(\mathtt{1}, \mathtt{2})$. By applying the exhaustive test, we found that $(\mathtt{1}, \mathtt{2})$, $(\mathtt{2}, \mathtt{4})$, $(\mathtt{3}, \mathtt{6})$ and their inverse are forming a group, which leads to $A = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\} \xleftrightarrow{S} A' = \{\mathtt{0}, \mathtt{2}, \mathtt{4}, \mathtt{6}\}$ in Table 4. Similarly, $(\mathtt{5}, \mathtt{7})$, $(\mathtt{a}, \mathtt{d})$, $(\mathtt{f}, \mathtt{a})$ and their inverse are forming a group, which leads to $A = \{\mathtt{0}, \mathtt{5}, \mathtt{a}, \mathtt{f}\} \xleftrightarrow{S} A' = \{\mathtt{0}, \mathtt{7}, \mathtt{a}, \mathtt{d}\}$ in Table 4.

In the end, all affine subspace transitions are recovered up to the vector space.

B Boolean Function Representation of S_4^{new} and S_5^{new}

S_4^{new} : `5 4 7 8 1 2 a 3 d b e 0 f c 6 9`

$$
\begin{aligned}
a' =&((a \text{ XOR } c) \text{ NAND } (b \text{ XOR } d)) \text{ NAND } (a \text{ NAND } (b \text{ XNOR } d))\\
b' =&((c \text{ NAND } d) \text{ NAND } (b \text{ NAND } (\text{ NOT } a))) \text{ NOR } (b \text{ NOR } (\ a \text{ NAND } d))\\
c' =&((\text{NOT } d) \text{ NOR } (a \text{ XOR } b)) \text{ NOR }\\
&((c \text{ XOR } d) \text{ NOR } ((\text{NOT } b) \text{ NOR } (a \text{ NOR } c)))\\
d' =&((a \text{ NOR } d) \text{ NOR } (c \text{ NAND } (\text{NOT } b))) \text{ NOR }\\
&((c \text{ XNOR } d) \text{ NOR } (b \text{ NOR } (a \text{ NOR } c)))
\end{aligned}
$$

$(\text{Depth}, \text{GEs})$ of the Boolean functions for a',b',c' and d' are $(4, 9)$, $(3.5, 6.5)$, $(4, 10)$ and $(4, 9.5)$, respectively.

S_5^{new} : `0 d 6 5 7 b e a 2 8 3 c 9 f 1 4`

$$
\begin{aligned}
a' =&((\text{ NOT } c) \text{ NOR } (a \text{ XOR } b)) \text{ NOR } (d \text{ NOR } ((\text{ NOT } b) \text{ NOR } (a \text{ NOR } c)))\\
b' =&((b \text{ OR } c) \text{ NAND } (a \text{ XNOR } d)) \text{ NAND } (d \text{ NAND } (a \text{ NOR } b))\\
c' =&(b \text{ NOR } (d \text{ NOR } (a \text{ NOR } c))) \text{ NOR } ((a \text{ NAND } b) \text{ NOR } (c \text{ NOR } (\text{ NOT } d)))\\
d' =&(d \text{ NAND } (a \text{ NOR } b)) \text{ NAND } ((c \text{ NOR } (\text{ NOT } b)) \text{ NOR } (d \text{ NOR } (a \text{ NAND } c)))
\end{aligned}
$$

$(\text{Depth}, \text{GEs})$ of the Boolean functions for a',b',c' and d' are $(4, 8)$, $(4, 7.5)$, $(4, 7.5)$ and $(4, 7.5)$, respectively.