Midori

Sankalp Acharya[1], Dakshraj Sadashiv Kashid[2] and Aditya Mishra[3]

[1] Indian Institute of Technology, Bhilai, India, sankalpacharya@iitbhilai.ac.in
[2] Indian Institute of Technology, Bhilai, India, dakshrajsadashiv@iitbhilai.ac.in
[3] Indian Institute of Technology, Bhilai, India, adityamishra@iitbhilai.ac.in

1 S-Box Analysis

Midori utilizes two types of bijective 4-bit S-Boxes, Sb_0 and Sb_1, where $Sb_0, Sb_1 : \{0,1\}^4 \rightarrow \{0,1\}^4$ (see Table 1). Sb_0 and Sb_1 are used in $Midori64$ and $Midori128$, respectively.

Table 1: 4−bit bijective S−boxes Sb_0 and Sb_1 in hexadecimal form

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Sb_0	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6
Sb_1	1	0	5	3	e	2	f	7	d	a	9	b	c	8	4	6

The code for this section is given in the file **S-Box Analysis.sagews**.

1.1 DDTs

A *difference distribution table* (DDT) is used to analyze the S-Box. The rows of the DDT encode the input difference and the columns encode the output difference. The cell $DDT[in, out]$ represents how often the input difference **in** give the output difference **out**. The DDTs for Sb_0 and Sb_1 can be generated using *SageMath*, as shown in Table 2.

Table 2: DDTs for Sb_0 and Sb_1, respectively

$\Delta_{in} \backslash \Delta_{out}$	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	0	2	2	2	0	2	0	0	0	0	0	2	0
2	0	4	0	0	4	0	0	0	0	4	0	0	4	0	0	0
3	0	0	0	0	2	0	4	2	2	2	0	0	0	2	0	2
4	0	2	4	2	2	2	0	0	2	0	0	2	0	0	0	0
5	0	2	0	0	2	0	0	4	0	2	4	0	2	0	0	0
6	0	2	0	4	0	0	0	2	2	0	0	0	2	2	0	2
7	0	0	0	2	0	4	2	0	0	0	0	2	0	4	2	0
8	0	2	0	2	2	0	2	0	0	2	0	2	2	0	2	0
9	0	0	4	2	0	2	0	0	2	2	0	2	2	0	0	0
a	0	0	0	0	0	4	0	0	0	0	4	0	0	4	0	4
b	0	0	0	0	2	0	0	2	2	2	0	4	0	2	0	2
c	0	0	4	0	0	2	2	0	2	2	0	0	2	0	2	0
d	0	0	0	2	0	0	2	4	0	0	4	2	0	0	2	0
e	0	2	0	0	0	0	0	2	2	0	0	0	2	2	4	2
f	0	0	0	2	0	0	2	0	0	0	4	2	0	0	2	4

$\Delta_{in} \backslash \Delta_{out}$	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	0	2	0	2	2	2	0	0	0	2	0	0	0
2	0	4	0	2	4	2	0	0	2	0	0	0	0	0	2	0
3	0	0	2	2	0	2	2	0	0	2	2	0	2	2	0	0
4	0	2	4	0	2	0	0	0	0	0	2	0	0	4	0	2
5	0	0	2	2	0	2	2	0	0	0	0	0	2	0	2	4
6	0	2	0	2	0	2	0	2	0	2	0	2	2	0	2	0
7	0	2	0	0	0	0	2	4	0	0	0	2	0	2	2	2
8	0	2	2	0	0	0	0	0	2	0	4	2	4	0	0	0
9	0	0	0	2	0	0	2	0	0	2	2	2	0	2	4	0
a	0	0	0	2	2	0	0	0	4	2	2	2	0	0	0	2
b	0	0	0	0	0	0	2	2	2	2	2	2	0	0	2	2
c	0	2	0	2	0	2	2	0	4	0	0	0	2	2	0	0
d	0	0	0	2	4	0	0	2	0	2	0	0	2	0	2	2
e	0	0	2	0	0	2	2	2	0	4	0	2	0	2	0	0
f	0	0	0	0	2	4	0	2	0	0	2	2	0	2	0	2

1.2 LATs

A *linear-approximation table* (LAT) is also used to analyze the S-Box. This table lists the probabilities that the sum of certain input bits of α equals the sum of certain output bits of $Sb_i[\alpha]$. Each entry gives us the *linear characteristic* for a pair of input-output masks: $\alpha \xrightarrow{S} \beta$ as well as the associated *bias*.
The LATs for Sb_0 and Sb_1 can be generated using *SageMath*, as shown in Table 3.

Table 3: LATs for Sb_0 and Sb_1, respectively

$\alpha\backslash Sb_0[\alpha]$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	4	2	-2	0	2	0	-2	0	2	0	4	-2	0	-2
2	0	4	0	0	4	0	0	0	-4	0	0	0	0	4	0	0
3	0	2	0	2	-2	0	2	4	2	-4	-2	0	0	2	0	2
4	0	-2	4	-2	2	0	-2	0	-2	-4	-2	0	0	-2	0	2
5	0	0	0	0	0	0	0	0	0	0	-4	-4	0	0	4	-4
6	0	2	0	2	-2	0	2	-4	-2	0	-2	0	-4	-2	0	2
7	0	0	0	4	0	0	-4	0	0	0	0	-4	0	0	-4	0
8	0	-2	-4	2	-2	0	-2	0	-4	-2	0	2	2	0	2	0
9	0	0	0	-4	-4	0	0	0	-2	2	-2	-2	2	2	-2	2
a	0	2	0	-2	-2	-4	-2	0	0	-2	4	-2	-2	0	2	0
b	0	0	0	0	0	-4	0	-4	2	-2	-2	2	2	2	-2	-2
c	0	4	0	0	0	0	-4	0	2	2	-2	2	2	-2	2	2
d	0	-2	4	2	-2	0	-2	0	0	2	0	2	-2	4	2	0
e	0	0	0	0	0	4	0	-4	2	-2	2	-2	2	2	2	2
f	0	-2	0	2	2	-4	2	0	0	2	0	-2	2	0	2	4

$\alpha\backslash Sb_1[\alpha]$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-2	4	-2	-4	-2	0	-2	-2	0	2	0	2	0	-2	0
2	0	4	2	-2	2	2	0	0	-2	-2	0	0	4	0	2	-2
3	0	-2	-2	0	-2	-4	0	2	0	-2	-2	0	2	0	4	-2
4	0	-4	2	-2	4	0	-2	2	0	0	2	2	0	0	2	2
5	0	-2	2	-4	0	2	2	0	2	0	-4	-2	-2	0	0	-2
6	0	0	0	0	-2	2	2	-2	2	2	2	2	0	-4	4	0
7	0	-2	0	2	2	0	-2	0	0	2	0	-2	2	-4	-2	-4
8	0	-2	-2	0	0	2	2	0	4	-2	2	0	4	2	-2	0
9	0	0	-2	-2	0	0	2	2	-2	-2	4	-4	-2	-2	0	0
a	0	2	0	-2	2	-4	2	0	2	4	2	0	0	2	0	-2
b	0	0	0	0	2	-2	2	-2	0	-4	0	4	-2	-2	-2	-2
c	0	2	4	2	0	-2	0	2	4	-2	0	-2	0	-2	0	2
d	0	0	0	0	0	0	-4	-4	2	-2	2	-2	-2	2	2	-2
e	0	-2	2	4	2	0	4	-2	-2	0	0	-2	0	2	2	0
f	0	0	-2	-2	2	-2	0	-4	0	0	-2	-2	2	-2	0	4

1.3 Differential-Uniformity

We now find the *differential-uniformity* of the S-Box. Differential-uniformity is nothing but the probability of the difference with the highest probability in absolute terms, i.e. how often it occurs in total.
It is calculated for Sb_0 and Sb_1 using *SageMath*, as shown below.

$$Du_{Sb_0} \to 4 \;\; \& \;\; Du_{Sb_1} \to 4$$

1.4 Differential Branch Number

The *differential branch number* is used to measure the diffusion power of a permutation, such as an S-Box. Let $wt(x)$ be the *Hamming weight* of a vector x. Hamming weight is nothing but the number of $1s$ in the vector. Also, let δ, Δ and $D_{Sb_i}[\delta, \Delta]$ be the input difference, output difference and the corresponding value in the DDT of the said S-Box. Then, the differential branch number β_{d_i} for Sb_i is:

$$\beta_{d_i} = \min_{\delta \neq 0, \Delta \neq 0, D_{Sb_i}[\delta,\Delta] \neq 0} \{wt(\delta) + wt(\Delta)\}$$

Thus, we get: $\beta_{d_0} \to 2$ & $\beta_{d_1} \to 2$.

1.5 Comparison with other Ciphers

Table 4: Comparing **Midori** with other similar ciphers

$Ciphers \backslash Parameters$	$S - Box\ Size(s)$	Du	β_d
$Midori$	$4 - bit$	4	2
$GIFT$	$4 - bit$	6	2
$Serpent$	$4 - bit$	4	3
$Prince$	$4 - bit$	4	2
$Pride$	$4 - bit$	4	2
$Ascon$	$5 - bit$	8	3
$Klein$	$4 - bit$	4	2
$PHOTON - Beetle$	$4 - bit$	4	3
LED	$4 - bit$	4	3
$Elephant$	$4 - bit$	4	3
$Wage$	$8 - bit$	8	2
$Aria$	$8 - bit$	4	2

2 Software Implementation

The implementation for $Midori$ has been done in **Python**. The code for the same is available in the zip file **Midori128-64-main.zip**. You can find the git-repo at Midori.

3 Construction

Midori is a family of two block ciphers: $Midori64$ and $Midori128$. Both ciphers accept $128 - bit$ keys, and have a different block size n ($n = 64$ for $Midori64$ and $n = 128$ for $Midori128$). The basic parameters of $Midori64$ and $Midori128$ are shown in Figure 1.

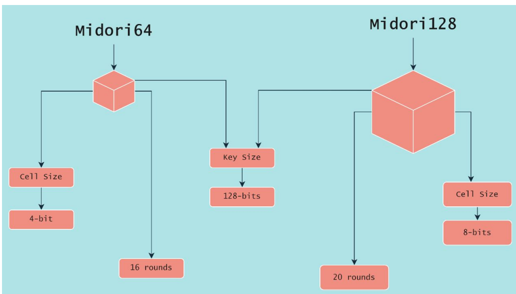

Figure 1: Parameters for $Midori64$ & $Midori128$

Midori is a variant of a Substitution Permutation Network (SPN), which consists of the S-layer and the P-layer, and uses a 4×4 matrix were the size of each cell are specified above.

3.1 S-Boxes & Matrices

Midori utilizes two types of bijective $4 - bit$ S-boxes, Sb_0 and Sb_1, where; Sb_0, Sb_1: $\{0,1\}^4 \rightarrow \{0,1\}^4$ (see Table 1). Sb_0 and Sb_1 are used in $Midori64$ and $Midori128$, respectively. Both the S-boxes have the **involution property**.

$Midori128$ utilizes four different $8-bit$ S-boxes SSb_0, SSb_1, SSb_2 & SSb_3, where; $SSb_0, SSb_1, SSb_2, SSb_3$: $\{0,1\}^8 \rightarrow \{0,1\}^8$. Mathematically, each SSb_i consists of input and output bit permutation (in_i and out_i) as shown in Figure 2. Each output bit permutation is taken as the inverse of the corresponding input bit permutation to keep the involution property.

$$in_0 : [0,1,2,3,4,5,6,7] \rightarrow [4,1,6,3,0,5,2,7]$$
$$in_1 : [0,1,2,3,4,5,6,7] \rightarrow [1,6,7,0,5,2,3,4]$$
$$in_2 : [0,1,2,3,4,5,6,7] \rightarrow [2,3,4,1,6,7,0,5]$$
$$in_3 : [0,1,2,3,4,5,6,7] \rightarrow [7,4,1,2,3,0,5,6]$$

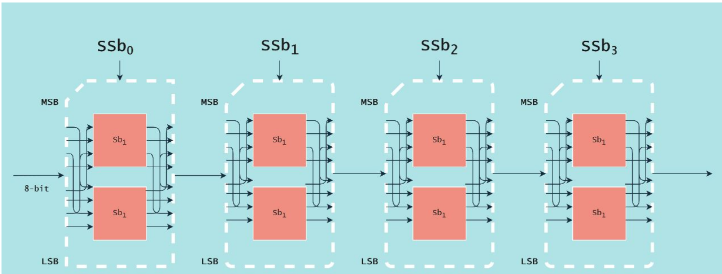

Figure 2: SSb_0, SSb_1, SSb_2 & SSb_3

Midori also utilizes an involutive binary matrix **M** defined as follows:

$$M_{4\times4} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

The matrix **M** updates $4m - bit$ values (a_0, a_1, a_2, a_3) using the following operation:

$$(a_0, a_1, a_2, a_3)^t \leftarrow M.(a_0, a_1, a_2, a_3)^t$$

3.2 Round Function

The round function of *Midori* consists of 3 layers: **S-Layer**, **P-Layer** and the **Key-Addition Layer**, as shown in Figure 3.

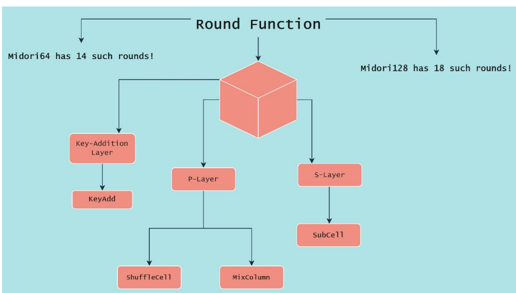

Figure 3: **Midori** Round Function

Each layer updates an $n - bit$ state **S** as follows:

1. **SubCell (S)**: Sb_0 and SSb_i are applied to every 4 and $8-bit$ cell of the state **S** of $Midori64$ and $Midori128$ in parallel, respectively. Namely, $s_i \leftarrow Sb_0[s_i]$ for $Midori64$ and $s_i \leftarrow SSb_{(i \ mod \ 4)}[s_i]$ for $Midori128$, where $0 \le i \le 15$.

2. **ShuffleCell (S)**: Each cell of the state is permuted as follows:

$$(s_0, s_1, \ldots, s_{15}) \leftarrow (s_0, s_{10}, s_5, s_{15}, s_{14}, s_4, s_{11}, s_1, s_9, s_3, s_{12}, s_6, s_7, s_{13}, s_2, s_8)$$

3. **MixColumn (S)**: **M** is applied to every $4m - bit$ column of the state **S**, i.e.,

$$(s_i, s_{i+1}, s_{i+2}, s_{i+3})^t \leftarrow M.(s_i, s_{i+1}, s_{i+2}, s_{i+3})^t \ \& \ i = 0, 4, 8, 12$$

4. **KeyAdd(S, RK_i)**: The $i-th \ \ n-bit$ round key RK_i is XORed to a state **S**.

3.3 Round Key Generation

For $Midori64$, a $128 - bit$ secret key **K** is denoted as two $64 - bit$ keys K_0 and K_1 as $K = K_0||K_1$. Then, $WK = K_0 \bigoplus K_1$ and $RK_i = K_{(i \ mod \ 2)} \bigoplus \alpha_i$, where $0 \le i \le 14$.

For $Midori128$, $WK = K$ and $RK_i = K \bigoplus \beta_i$, where $0 \le i \le 18$. It can be seen that the constants α_i and β_i are in the form of 4×4 binary matrices in Table 5. They are added bit-wise to the LSB of every round key byte in $Midori128$ and round key nibble in $Midori64$ respectively. Note that $\alpha_i = \beta_i$ for $0 \le i \le 14$.

Table 5: The Round Constants β_i

i	β_i	i	β_i	i	β_i	i	β_i	i	β_i	i	β_i	i	β_i
0	$\begin{matrix}0&0&1&0\\0&1&0&0\\0&0&1&1\\1&1&1&1\end{matrix}$	1	$\begin{matrix}0&1&1&0\\1&0&1&0\\1&0&0&0\\1&0&0&0\end{matrix}$	2	$\begin{matrix}1&0&0&0\\0&1&0&1\\1&0&1&0\\0&0&1&1\end{matrix}$	3	$\begin{matrix}0&0&0&0\\1&0&0&0\\1&1&0&1\\0&0&1&1\end{matrix}$	4	$\begin{matrix}0&0&0&1\\0&0&1&1\\0&0&0&1\\1&0&0&1\end{matrix}$	5	$\begin{matrix}1&0&0&0\\1&0&1&0\\0&0&1&0\\1&1&1&0\end{matrix}$	6	$\begin{matrix}0&0&0&0\\0&0&1&1\\0&1&1&1\\0&0&0&0\end{matrix}$
7	$\begin{matrix}0&1&1&1\\0&0&1&1\\0&1&0&0\\0&1&0&0\end{matrix}$	8	$\begin{matrix}1&0&1&0\\0&1&0&0\\0&0&0&0\\1&0&0&1\end{matrix}$	9	$\begin{matrix}0&0&1&1\\1&0&0&0\\0&0&1&0\\0&0&1&0\end{matrix}$	10	$\begin{matrix}0&0&1&0\\1&0&0&1\\1&0&0&1\\1&1&1&1\end{matrix}$	11	$\begin{matrix}0&0&1&1\\0&0&0&1\\1&1&0&1\\0&0&0&0\end{matrix}$	12	$\begin{matrix}0&0&0&0\\1&0&0&0\\0&0&1&0\\1&1&1&0\end{matrix}$	13	$\begin{matrix}1&1&1&1\\1&0&1&0\\1&0&0&1\\1&0&0&0\end{matrix}$
14	$\begin{matrix}1&1&1&0\\1&1&0&0\\0&1&0&0\\1&1&1&0\end{matrix}$	15	$\begin{matrix}0&1&1&0\\1&1&0&0\\1&0&0&0\\1&0&0&1\end{matrix}$	16	$\begin{matrix}0&1&0&0\\0&1&0&1\\0&0&1&0\\1&0&0&0\end{matrix}$	17	$\begin{matrix}0&0&1&0\\0&0&0&1\\1&1&1&0\\0&1&1&0\end{matrix}$	18	$\begin{matrix}0&0&1&1\\1&0&0&0\\1&1&0&1\\0&0&0&0\end{matrix}$				

An interesting fact is that the constants have been derived from the hexadecimal encoding of the fractional part of $\pi = 3.243f \ 6a88 \ 85a3 \ \ldots$ For example, the 1st, 2nd, 3rd, 4th rows of β_0 when read as a $4 - bit$ binary constant, are the encoding of the hex values $2, 4, 3, f$ respectively.

3.4 Overview

The Figure below shows the overview of **Midori64**. It can be similarly shown for **Midori128**, just by having 20 rounds instead of the 16 rounds.

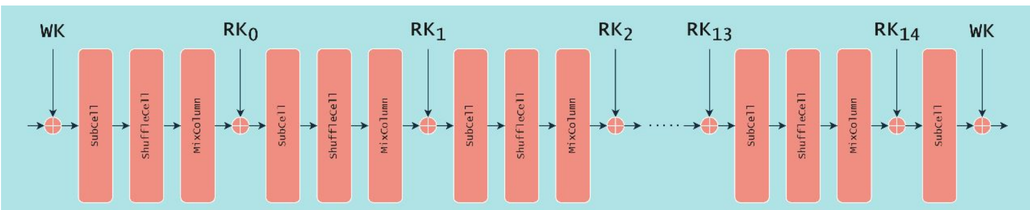

Figure 4: Overview of **Midori64**

4 General Cryptanalysis

4.1 Differential Cryptanalysis

It is a general form of cryptanalysis applicable primarily to block ciphers. In essence, it is the analysis of how input differences can affect the resultant out differences to recover the secret key.
We calculated **Differential Uniformity** above to be 4 for our S-Boxes. We can find the maximum differential probability of the S-Boxes using the formula below:

$$MDP_{Sb_i} = \frac{Du}{2^4} \implies MDP_{Sb_i} = 2^{-2}$$

Thus, the maximum differential and linear probabilities of $Sb_0, SSb_0, SSb_1, SSb_2$ & $SSb_3 \rightarrow 2^{-2}$, respectively.
We found out the minimum number of differentially active S-Boxes of **Midori64** and **Midori128** shown in Table 6.

Table 6: Minimum Number of differentially active S-Boxes

Round Number (r)	4	5	6	7	8	9	10	11	12	13	14	15	16
Min. of *diff.* active S-Boxes	16	23	30	35	38	41	50	57	62	67	72	75	84

$Midori64$ and $Midori128$ have more than **32** and **64** active S-boxes after 7 and 13 rounds. Thus, we expect that variants of $Midori64$ and $Midori128$ reduced to 7 rounds and 13 rounds do not have any differential and linear trails whose probabilities are higher than 2^{-64} and 2^{-128}.

4.2 Integral Cryptanalysis

We tried to find actual integral characteristics, but failed to do so. By exploiting several techniques used in the integral attacks, we can construct 7-round key recovery attacks based on the distinguisher but more round seems to be infeasible. But, we were unable to find any such attacks. Thus, full versions of **Midori64** and **Midori128** are expected to be enough secure against integral attacks.

5 Automated Cryptanalysis

The differential cryptanalysis can be modeled as **MILP** (Mixed Integer Linear Programming). It can be solved using MILP solver to get the differential trail for the attack. In case of the block cipher like our the main aim is to find the minimum number of active S-boxes, which will give the required input difference and trail for the attack.

The paper mention the same analysis and the result is given in Table 6. MILP model for one round:

In single round, except the round key addition other operation will affect the input and output difference.

1. **SubCell**: If a input bit is active then Sbox must be active.

$$S - x_i \geq 0 \;\; i \in 0, 1, 2, 3$$

And if a S-box is active one of its input bit is also active

$$\sum_{i=0}^{4} x_i - S \geq 0$$

.

2. **ShuffleCell**: In shuffle cell, each block is mapped to different block. Consider $B_1 \longrightarrow B_{10}$ to maintain this we introduces inequality for each bit

$$B_1 x_i - B_{10} x_i = 0$$

$$M_{4\times4} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

3. **MixColumn**: As each block is affected by the rest of the three blocks in its column.

$$B_0' = B_4 + B_8 + B_{12}$$

6 Software Application using Midori

We have implemented a **UDP Client Server** using $Midori128$. UDP (User Datagram Protocol) is a transport layer protocol which is unreliable as well as connectionless. Hence, we apply **Midori** to ensure security. A secret key is shared between the users. We use $Midori$ as it is especially practical in restrained environments, and complements the quick speed of UDP. The code for the same is available in **App.zip**. You can find the git-repo at Midori.