Gift Cipher

Term paper by Team 3-cliqued

Swetha Reddy[1](11940250)
Tarun Singh[2](11941230)
Charanesh raj[3](11940300)

Indian Institute of Technology Bhilai, bathalas@iitbhilai.ac.in
Indian Institute of Technology Bhilai, taruns@iitbhilai.ac.in
Indian Institute of Technology Bhilai, charaneshr@iitbhilai.ac.in

Abstract. In this paper, we have provided a full description of Gift Cipher. GIFT is a light-weight cipher with a strong reputation for its software and hardware implementation. We drew motivation and provided detailed implementation of Gift Cipher after the Present Cipher. The number of encryption rounds and the actions required in each round are clearly stated. We also worked on their security analysis and tested it against various threats. We also reviewed on how to determine the maximum number of active Sboxes. Differential and Linear Cryptanalysis' data, time, and memory challenges were also examined.

Keywords: GIFT-64, rounds, Sbox, roundKey, Differential, Integral, Security, Analysis, DDT, LAT, Complexity, XOR, permutation, Hull Effect

1 Introduction

Following the publication of PRESENT ten years ago, the GIFT Cipher, a lightweight block cipher that exceeds PRESENT in terms of efficiency and security, is unveiled. PRESENT's known weak point is Linear hulls and it's diffusion layer is merely a bit permutation and the Sbox is responsible for the majority of the security. Although this Sbox has outstanding cryptographic features, it is rather expensive. So, by removing the constraint of having single active Sbox, GIFT has much cheaper Sbox. The design of GIFT is clean and elegant. It saves a lost of energy when compared with ciphers currently available. It approaches a point when storage and Sboxes occupy nearly the whole deploy area, and any inexpensive Sbox option results in a very poor approach. GIFT is nothing more than Sbox with bit-wiring, but the flow of its bitslice infomration provides great performance in all situations, from area-optimized embedded systems to exceptionally fast software applications on high-end platforms.

2 Specifications

At present GIFT-64 and GIFT-128 are available which have a 28-round and 40-round SPN cipher respectively, both with a 128-bit key.GIFT can be interpreted in three ways. In this study, we describe the bits in a row using the usual 1D format, such as PRESENT. It may alternatively be defined in terms of bitslice 2D, a rectangular array like RECTANGLE, and so on.

3 Round Function:

Each GIFT round consists of three steps: Subcells, PermBits, and AddRoundKey, which is similar in concept to wrapping a gift.:

1. Embed the information in a box (Substituition Cells)
2. Tie the ribbon around the box and secure it with a bow (Permutation Bits)
3. To secure the content, tie a knot (Round Key Addition).

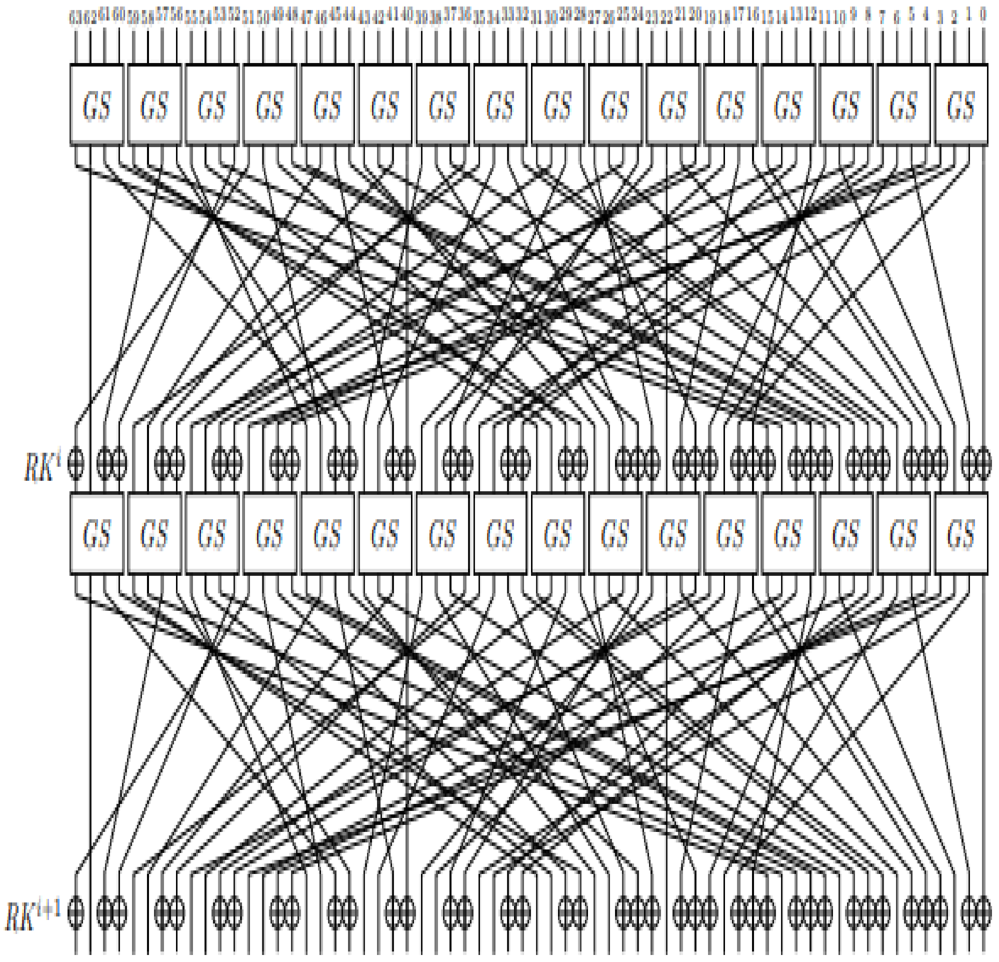


Fig 1. Two Rounds of GIFT-64

4 Initialization:

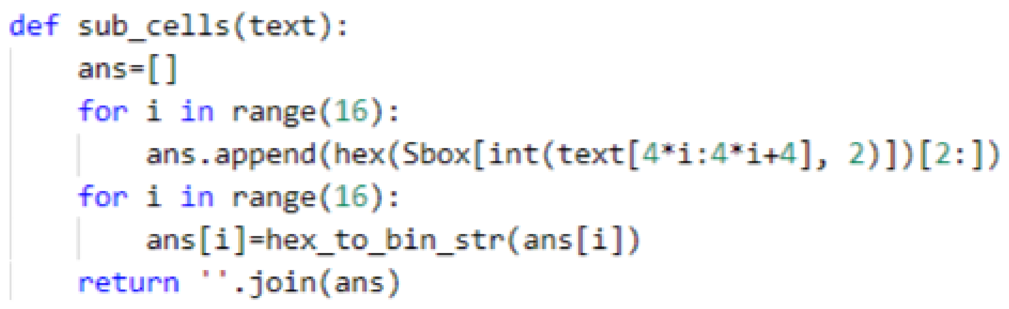
From the cipher state S, the cipher gets an n-bit plaintext b_{n-1}, b_{n-2},... b_0, where n = 64, 128 and b_0 is the least significant bit. The cipher state can also be expressed as 16 or 32, 4-bit nibbles where S is, $S = w_{s1}||w_{s2}||...||w_0$. The cipher additionally gets a 128-bit key K as the key state, where k_i is a 16-bit string.

1. **SubCells**

GIFT makes use of a 4-bit invertible Sbox. The Substituition box is applied to each cipher state bit.
Sbox used for GIFT implementation,

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

Fig 2. SBox of GIFT-64



```
def sub_cells(text):
    ans=[]
    for i in range(16):
        ans.append(hex(Sbox[int(text[4*i:4*i+4], 2)])[2:])
    for i in range(16):
        ans[i]=hex_to_bin_str(ans[i])
    return ''.join(ans)
```

Fig 3. Code for SubCells Implementation

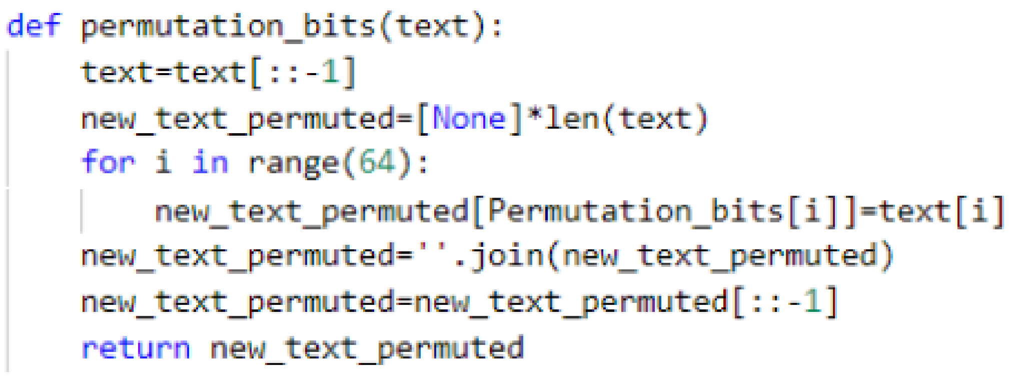
2. **PermBits**

The GIFT-64 bit permutation has the unique characteristic that each bit situated in a segment remains in the same segment after this permutation.
The permutations of GIFT-64 are calculated using the algorithm below:

$P_{64}(i) = 4\lfloor \frac{i}{16} \rfloor + 16((3\lfloor(\text{i mod}16)/4\rfloor + (\text{i mod}4))\ \text{mod}4) + (\text{i mod}4)$

PermBits																
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P_0(i)	0	12	8	4	1	13	9	5	2	14	10	6	3	15	11	7
P_1(i)	4	0	12	8	5	1	13	9	6	2	14	10	7	3	15	11
P_2(i)	8	4	0	12	9	5	1	13	10	6	2	14	11	7	3	15

Fig 4. Table showing PermBits in the first 3 rounds of Encryption

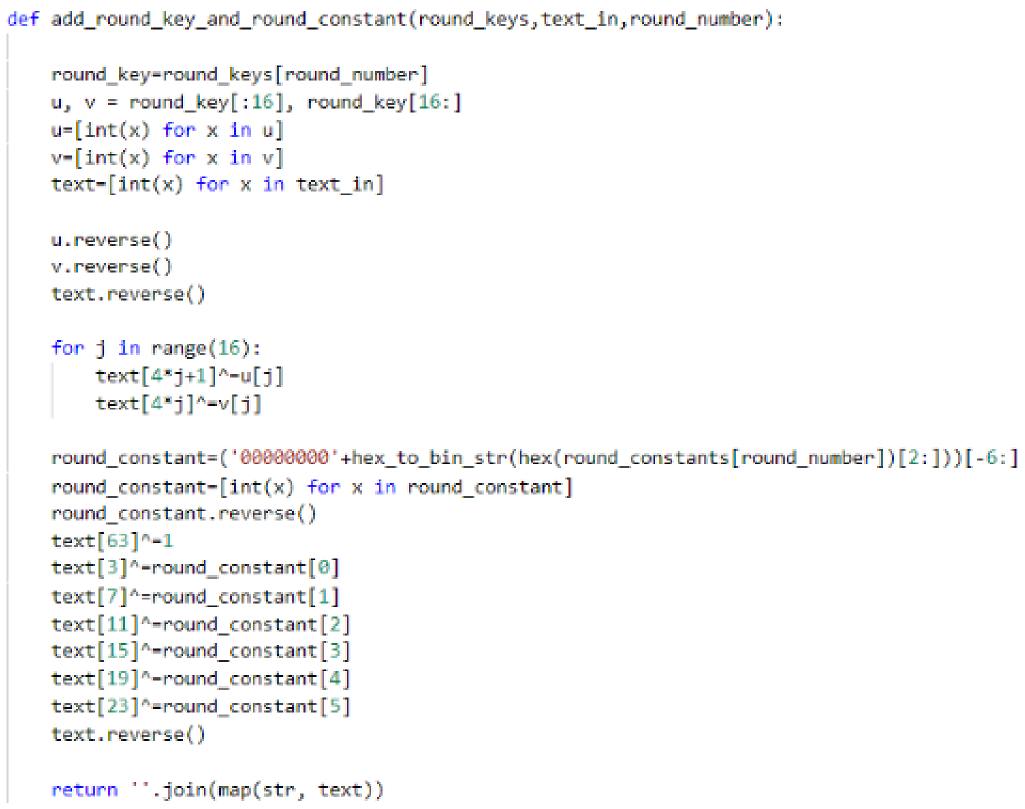


```
def permutation_bits(text):
    text=text[::-1]
    new_text_permuted=[None]*len(text)
    for i in range(64):
        new_text_permuted[Permutation_bits[i]]=text[i]
    new_text_permuted=''.join(new_text_permuted)
    new_text_permuted=new_text_permuted[::-1]
    return new_text_permuted
```

Fig 5. Code for Permutation of Bits

3. **AddRoundKey**
The round key and round constant are added in this phase. As the round key, two 16-bit segments U, V are recovered from the key state. Thereafter, for GIFT-64 round key addition, U and V are XORed to S_1 and S_0 of the cipher state, respectively.

$$S_1 \longleftarrow S_1 \oplus U$$
$$S_0 \longleftarrow S_0 \oplus V$$



```
def add_round_key_and_round_constant(round_keys,text_in,round_number):

    round_key=round_keys[round_number]
    u, v = round_key[:16], round_key[16:]
    u=[int(x) for x in u]
    v=[int(x) for x in v]
    text=[int(x) for x in text_in]

    u.reverse()
    v.reverse()
    text.reverse()

    for j in range(16):
        text[4*j+1]^=u[j]
        text[4*j]^=v[j]

    round_constant=('00000000'+hex_to_bin_str(hex(round_constants[round_number])[2:]))[-6:]
    round_constant=[int(x) for x in round_constant]
    round_constant.reverse()
    text[63]^=1
    text[3]^=round_constant[0]
    text[7]^=round_constant[1]
    text[11]^=round_constant[2]
    text[15]^=round_constant[3]
    text[19]^=round_constant[4]
    text[23]^=round_constant[5]
    text.reverse()

    return ''.join(map(str, text))
```

Fig 6. Code for Round Key addition

Round Constants and Key Schedule
The sole change between the two versions of GIFT(64, 128) is the round key extraction, which uses a different round constants and key schedule. Before updating the key state, a round key is taken from the key state. Here the round key is RK = $U||V$

$$U \longleftarrow K_1,\, V \longleftarrow K_0$$

key-state is now updated as follows,

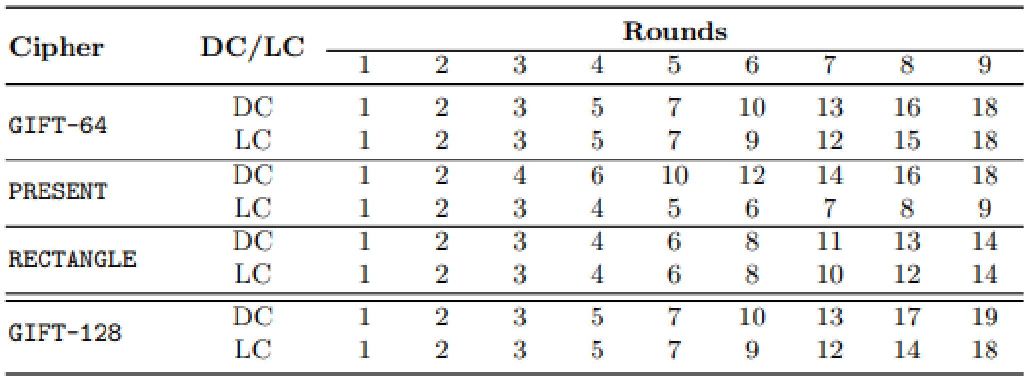
$$K_7||K_6||....K_1||K_0 \longleftarrow K_1 >> ||K_0 >> 12||....K_3||K_2$$

The round constants are created using a 6-bit LFSR with the state $r_5 r_4 r_3 r_2 r_1 r_0$. It has the following update function:

$$(cr5, r_4, r_3, r_2, r_1, r_0) \longleftarrow (r_4, r_3, r_2, r_1, r_0, r_5 \oplus r_4 \oplus 1)$$

5 Security Analysis

For block ciphers,linear and differential cryptanalysis are two of the most powerful approaches known. The most common and fundamental security analysis is determining a cipher's resistance to differential and linear cryptanalysis. One method of determining a cipher's resistance is to find the lower limit for the number of active Sboxes involved in a differential or linear characteristic.



Cipher	DC/LC	Rounds								
		1	2	3	4	5	6	7	8	9
GIFT-64	DC	1	2	3	5	7	10	13	16	18
	LC	1	2	3	5	7	9	12	15	18
PRESENT	DC	1	2	4	6	10	12	14	16	18
	LC	1	2	3	4	5	6	7	8	9
RECTANGLE	DC	1	2	3	4	6	8	11	13	14
	LC	1	2	3	4	6	8	10	12	14
GIFT-128	DC	1	2	3	5	7	10	13	17	19
	LC	1	2	3	5	7	9	12	14	18

Fig 7. Limits of different Ciphers on number of Active SBoxes

These goals were met after 9 rounds of GIFT. As a result, the focus of our DC and LC research and debate is on 9-round.

6 Differential Cryptanalysis

The differential probability can be calculated by adding the probabilities of all differential features that have the same inlet and outlet variations.We first develop a differential characteristic with the fewest active Sboxes to compute a 9-round differential probability of GIFT. After that, we sum up the probabilities by setting the inlet and outlet variances and searching for the next best feasible differential feature. The search ends when the next differential characteristic makes a negligible contribution to further enhancing the differential probability.

The differential probability of GIFT-64 is lower than 2^{-63} when we have 14-round since it has a 9-round differential probability of $2^{-44.415}$. If we take an average for each round and transmit forward, we estimate the differential probability to be lower than 2^{-63} when we have 14-round. As a result, we feel that the 28-round GIFT-64 is sufficient to fend off differential cryptanalysis.

It's worth noting that while an ideal differential feature has the fewest active Sboxes in most circumstances, following differential characteristics with the same input and output variations have much more active Sboxes than the initial feature. As a result, the differential probability is near to the ideal differential characteristic's probability. With the exception of PRESENT, which has various optimum differential characteristics for some fixed source and target differences due to its symmetrical structure.

GIFT-64 Data Complexity $\longrightarrow 2^{63}$
GIFT-64 Time Complexity $\longrightarrow 2^{112}$
GIFT-64 Memory Complexity $\longrightarrow 2^{80}$

7 Linear Cryptanalysis

We discover an ideal linear characteristic initially, then fix the inlet and outlet masks to get the next highest suitable linear characteristic and total the correlated potentials, similar to differential. When a succeeding linear characteristic contributes insignificantly to the linear hull effect, the search is ended.

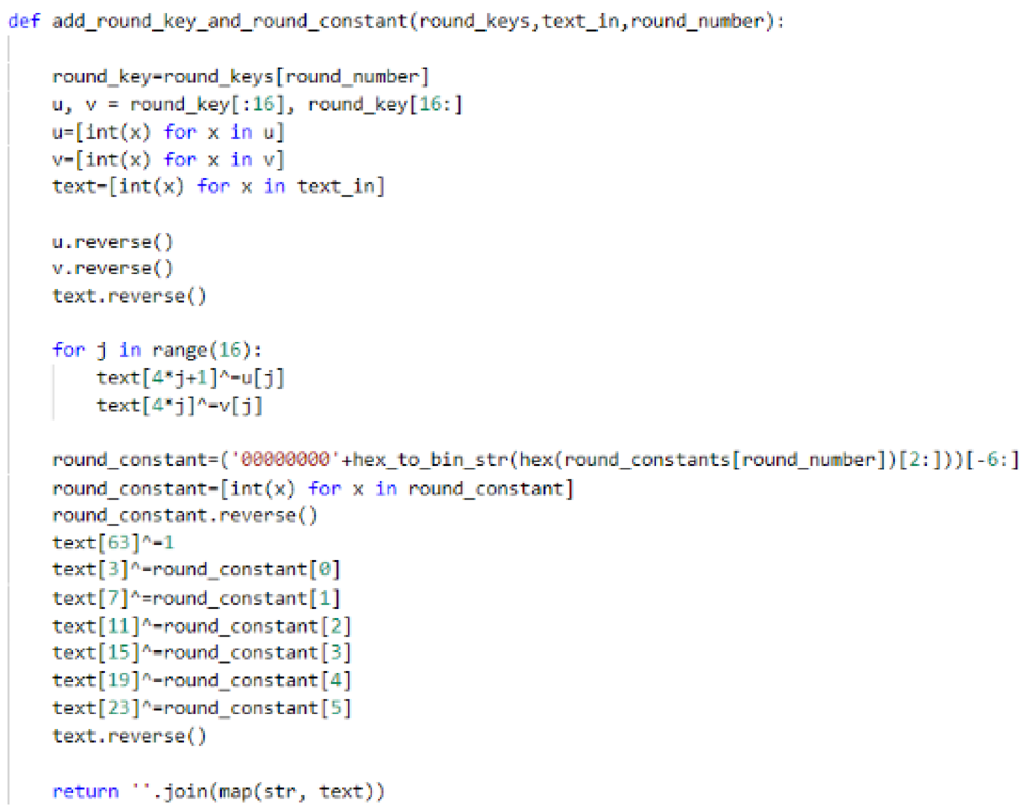
The Hull Effect - The total of the correlation potentials of all linear trails between input and output selection patterns is the average correlation potential between input and output selected patterns.
GIFT-64 has a 9-round linear hull effect of $2^{-49.996}$, and it is predicted to need 13 rounds to attain a correlation potential less than 2^{-64}. As a result, we feel that 28 rounds of GIFT-64 are sufficient to withstand Linear cryptanalysis.

GIFT-128 has a 9-round linear hull effect of 2−45.99, implying that we would need roughly 27 rounds to attain a correlation potential less than 2−128. As a result, we anticipate 40 rounds of GIFT-128 will be sufficient to withstand linear cryptanalysis .
GIFT-64 Data Complexity $\longrightarrow 2^{63}$
GIFT-64 Time Complexity $\longrightarrow 2^{96}$
GIFT-64 Memory Complexity $\longrightarrow 2^{63}$

8 Related-Key Differential Cryptanalysis

Because it takes four rounds for all the keys which are going to cipher state in GIFT-64, it's easy to see how there can be no active Sboxes from one to four rounds. As a result, from round 5 onwards, we begin computing the related-key differential bounds.These differential characteristics have a probability of $2^{1.415}, 2^{5}, 2^{6.415}, 2^{10}, 2^{16}, 2^{22}, 2^{27}, 2^{33}$ from round 5 to round 12.

GIFT-64 uses 32-bit round keys that are derived from the 128-bit main key. For all the 128-bits of the main key to be involved, it will take at-least 128/32 = 4 rounds. Thus, we could say it is possible that there may not be any active S-box from round 1 to round 4. So we need to begin with related key cryptanalysis-based calculations from round 5 onwards. If we look at the probabilities of the differential characteristics, we may observe that the probability of 12-round characteristic has lower bound of 2^{-33}. Hence, 28 rounds might not be very secure against related key cryptanalysis. We could increase the number of rounds in our implementation to reduce the probability of of such attacks. We can see here that key state is only XOR with only half of the text at some chosen points and not with all the bits. Also, the XOR is happening with the same bits every round. This also increases the efficiency.
Each bit of the 6-bit round constant is XOR with a different nibble in the text to reduce any symmetry.



```python
def add_round_key_and_round_constant(round_keys,text_in,round_number):

    round_key=round_keys[round_number]
    u, v = round_key[:16], round_key[16:]
    u=[int(x) for x in u]
    v=[int(x) for x in v]
    text=[int(x) for x in text_in]

    u.reverse()
    v.reverse()
    text.reverse()

    for j in range(16):
        text[4*j+1]^=u[j]
        text[4*j]^=v[j]

    round_constant=('00000000'+hex_to_bin_str(hex(round_constants[round_number])[2:]))[-6:]
    round_constant=[int(x) for x in round_constant]
    round_constant.reverse()
    text[63]^=1
    text[3]^=round_constant[0]
    text[7]^=round_constant[1]
    text[11]^=round_constant[2]
    text[15]^=round_constant[3]
    text[19]^=round_constant[4]
    text[23]^=round_constant[5]
    text.reverse()

    return ''.join(map(str, text))
```

Fig 8. Observations

9 Software Implementation

It appears logical to assume that the most efficient software implementations of GIFT will be bitslice implementations due to its intrinsic bitslice design.
As a result, when you're working with 64-bit items, the initial step is to compress the data into bitslice form. We'll need to keep at least four words filled with original since GIFT uses a 4bit Sbox. We want to use eight of these registers since we can utilise the powerful pshufb instruction to achieve bit permutation. Our method will encrypt 8y blocks at each cycle if one word can hold y blocks of GIFT information.

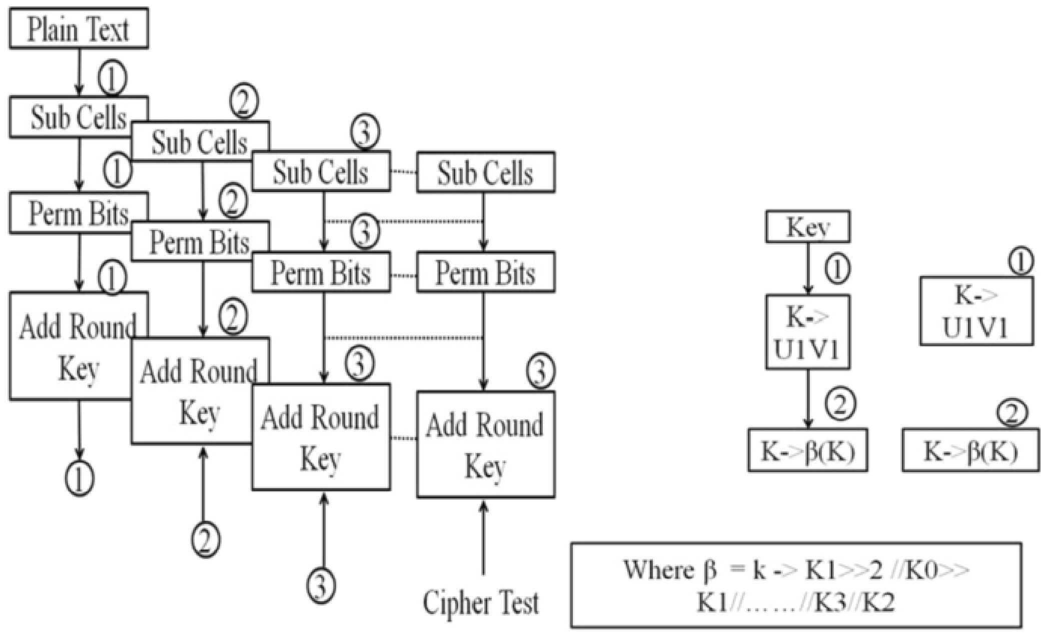


Fig 9. GIFT Cipher Implementation

Packing and Unpacking: First, all sixteen 64-bit blocks are loaded into the 8 128-bit registers.

$$R_1 = b^1_{63}...b^1_1b^1_0||b^0_{63}...b^0_1b^0_0$$
$$R_3 = b^3_{63}...b^3_1b^3_0||b^2_{63}...b^2_1b^2_0$$
$$..$$
$$R_8 = b^{15}_{63}...b^{15}_1b^{15}_0||b^{14}_{63}...b^{14}_1b^{14}_0$$

Using the SWAP-MOVE technique, packing and unpacking may be done quickly and easily.
SWAP-MOVE(C, D, M, N):
T = ((C >> N) $\oplus D)M$
$D = D \oplus Tem$
$C = C \oplus (T«N)$

Using only six logical operations, this procedure will exchange the bits in D masked by M with the bits in C masked by $(M << N).Asaresult, alloftheSboxes'1^{st}$, $2^{nd}, 3^{rd}, and 4^{th}$ bits may be packed into $R_4/R_8, R_3/R_7, R_2/R_6$ and R_1/R_5, using:
SWAP-MOVE(R_1, R_2, 0xaa...aaa, 1)
SWAP-MOVE(R_5, R_6, 0xaa...aaa, 1)
SWAP-MOVE(R_3, R_4, 0xaa...aaa, 1)
SWAP-MOVE(R_7, R_8, 0xaa...aaa, 1)
SWAP-MOVE(R_1, R_3, 0xaa...aaa, 1)
SWAP-MOVE(R_5, R_7, 0xaa...aaa, 1)
SWAP-MOVE(R_2, R_4, 0xaa...aaa, 1)
SWAP-MOVE(R_6, R_8, 0xaa...aaa, 1)

After that, we may aggregate these snatches into bytes:
SWAP-MOVE(R_1, R_5, 0xf0f0...0f0, 4)
SWAP-MOVE(R_2, R_6, 0xf0f0...0f0, 4)
SWAP-MOVE(R_3, R_7, 0xf0f0...0f0, 4)
SWAP-MOVE(R_4, R_8, 0xf0f0...0f0, 4)

The bits are now arranged into packets of bytes as desired, but bits within the same plain text message are distributed across many registers, decelerating the implementation. We use a few unpacking commands like punpckhbw and punpcklbw to group them together.

Round Function :
Computing the round function is simple once the information is in bitslice mode. The GIFT Sbox is implemented in a software-optimized bitslice which uses just 1 OR, 1 NOT, 3 ANDs, and , 6 XORs,instruction.
Now that the information is compressed into bytes, performing the bit permutation is similarly straightforward. However, one of the most important properties of the GIFT bit permutations is that a bit in segment I is always transmitted to a ames slice I during the process. As a result, using the bit alternation layer essentially implies randomly permuting the sequence of the bytes within the registers. With just one pshufb command for each register, the whole bit permutation may be executed.
Subkey addition is performed on the state's 1^{st} and 2^{nd} bits. Because numerous plain text blocks are frequently ciphered under the same keys, it appears to be a smart method to precompute all the round substituition keys, save them, and return them with only one memory visit when needed. The key schedule can also be completed quickly by utilising the pshufb command to finish the key permutation.

10 Conclusion

Software bitsliced solutions can be extremely efficient because of the GIFT fixsliced representation. Except for SPECK-64/128, GIFT-64 has given better performance when compared with all other 64-bit ciphers. The GIFT-128 performs 1.6 times better than the GIFT-64. GIFT-128 implementations outperform the existing AES-128 standard by a wide margin. When compared to AES-128, GIFTb-128 saves roughly 28 percent of the cycles. MILP based Cryptanalysis is far more efficient when compared with the classical way of implementing Differential and Linear Cryptanalysis.