PRINT Cipher

Manohar Lal Das[1], Aman Khan[2] and Aayush Deshmukh[3]

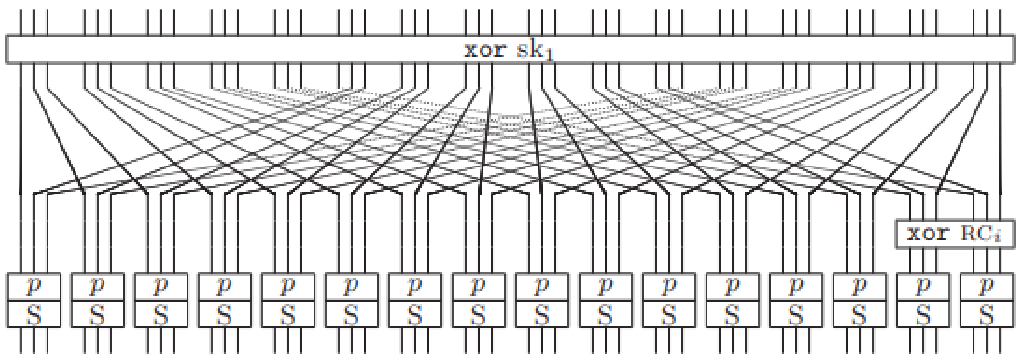
[1] IIT Bhilai, Raipur, India, manoharlal@iitbhilai.ac.in
[2] IIT Bhilai, Raipur, India, amankhan@iitbhilai.ac.in
[3] IIT Bhilai, Raipur, India, aayushd@iitbhilai.ac.in

Abstract. Print Cipher is one of the lightweight SPN network with 48-bit and 96-bit block cipher for IC-printing. It is design to make use of the properties of IC-printing technology. Print cipher is still in the beginning phase of their development but allow the production of different circuits at low cost.

Keywords: SPN · IC-printing

1 Introduction

In order to spot items using smart bar-codes, we use RFID tags and sensors, the safety of contrived hardware devices like RFID tags is a major concern in cryptography nowadays. PRINT Cipher, which was introduced in two forms a 48-bit and another 96-bit cipher, was published in "Cryptographic Hardware and Embedded systems (CHES 2010)". This cipher can have either 80 or 160 bit unrevealed keys. The essential and appealing properties of PRINTcipher are that each one round uses the identical round key and differs only by a round counter in which the linear layer is partially key-dependent. Invariance subspace attacks are promising attack results on PRINT Cipher on the whole Printcipher- 48/96. Here the supported weak keys are considered as the very well known Cryptanalytic result, and the simplest attack encountered is invariance subspace attacks on the complete PRINTcipher - 48/96. It requires 5 choosen plaintexts that are applicable to PRINT Cipher- 48 attacks which has 2^{53} weak keys, and whose computational complexity is negligible. Similarly 2^{102} Weak Key is there in PRINT Cipher- 96 and the attack requires five chosen plaintexts whose computational complexity is negligible.



2 Main Result

2.1 Sbox Analysis

The sbox for the PRINT cipher is a 3-bit to 3-bit. Since input is 3-bit so for a b-bit block, the sbox is applied $\frac{b}{3}$ parallely. The current state for the sbox is a $\frac{b}{3}$ words, for each word same sbox is used and concatinating the output gives the next state.It is a balanced sbox and has a linear structure.The sbox is shown below :-

x	0	1	2	3	4	5	6	7
s[x]	0	1	3	6	7	4	5	2

2.1.1 Difference Distribution Table

The sbox has a differential branch number defined as $\min_{v,\, w \neq v} \{wt(v \oplus w) + wt(S(v) \oplus S(w))\}$ of **2**. The difference distribution table (ddt) which is generated using Sage is as follows :-

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	2	0	2	0	2	0	2
2	0	0	2	2	0	0	2	2
3	0	2	2	0	0	2	2	0
4	0	0	0	0	2	2	2	2
5	0	2	0	2	2	0	2	0
6	0	0	2	2	2	2	0	0
7	0	2	2	0	2	0	0	2

2.1.2 Linear Approximation Table

The linear branch number which is denoted by $\min_{\alpha \neq,\, \beta,\, LAM(a,b) \neq 0}\{wt(a) + wt(b)\}$ for this sbox is **2**. The linearity of this sbox is **4**. The linear approximation table generated from Sage is as follows:-

	0	1	2	3	4	5	6	7
0	4	0	0	0	0	0	0	0
1	0	-2	0	2	0	2	0	2
2	0	0	2	2	0	0	2	-2
3	0	2	-2	0	0	2	2	0
4	0	0	0	0	2	-2	2	2
5	0	2	0	2	2	0	-2	0
6	0	0	2	-2	2	2	0	0
7	0	2	2	0	-2	0	0	2

2.1.3 Additional Properties of Sbox

1. The component funcion in 3 variables in algebraic normal form of the sbox is

x0*x2 + x0 + x1*x2

2. The interpolation polynomial for the sbox is

$$\mathbf{(a + 1)x^6 + (a^2 + a + 1)x^5 + (a^2 + 1)x^3}$$

3. The polynomials which satisfy the sbox are :-

- x0*x2 + x0 + x1 + y1
- x0*x1 + x0 + x1 + x2 + y2
- x0*y1 + x0 + x2+ y1 + y2
- x0*y2 + x1 + y1
- x1*x2 + x0 + y0
- x1*y0 + x1 + x2 + y0 + y2
- x0*y0 + x1*y1 + x2 + y2
- x1*y2 + x0 + x1 + y0
- x2*y0 + x1 + y0 + y1
- x2*y1 + x0 + y0
- x0*y0 + x2*y2 + x0 + x1 + x2 + y0 + y1
- y0*y1 + x2 + y0 + y1 + y2
- y0*y2 + x1 + y1
- y1*y2 + x0+ y0 + y1

x - input variables y - output variables

4. Maximum degree of component function - 2

5. Minimum degree of component function - 2

6. Maximal differential probability - 0.25

7. Absolute maximal linear bias - 2

8. Relative maximal linear bias - 0.25

2.2 Cryptanalysis of PRINT Cipher

While in related-key attacks on PRINTcipher-48/96 to find the weakness. On key-dependent permutation part related key have different values. By constructing related key of t-round the differential characteristics is of probability 2^{-t} so we will be able to extract the secret keys used PRINTcipher-48/96.

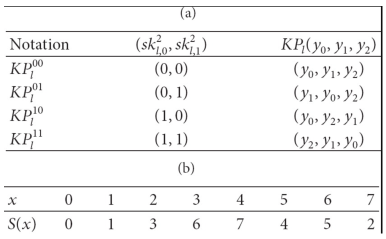
To get back 80-bit secret key of PRINTcipher-48 it need 4 related key PRINTcipher-48 need 2^{47} related-key chosen plaintexts with complexity of $2^{60.62}$ To get back 160-bit secret key of PRINTcipher-96 it need 4 related key PRINTcipher-96 need 2^{47} related-key chosen plaintexts with complexity of 2^{107}

2.3 Construction of Related-Key Differential Characteristics on PRINTcipher

Using attributes of a key-dependent permutation KP and S-box, steps to produce t round related-key differential characteristics on printcipher

2.3.1 S-Box and Related-Key Properties on Key-Dependent Permutation :-

A key-dependent permutation KP_l with a 3-bit input value (y0, y1, y2). Round-key of 2-bit is is equal to (0,0) or (0,1)



(a)

Notation	$(sk^2_{l,0}, sk^2_{l,1})$	$KP_l(y_0, y_1, y_2)$
KP_l^{00}	(0, 0)	(y_0, y_1, y_2)
KP_l^{01}	(0, 1)	(y_1, y_0, y_2)
KP_l^{10}	(1, 0)	(y_0, y_2, y_1)
KP_l^{11}	(1, 1)	(y_2, y_1, y_0)

(b)

x	0	1	2	3	4	5	6	7
$S(x)$	0	1	3	6	7	4	5	2

corresponding output value is computed as follows:

$$(0,0): (y_0, y_1, y_2) \rightarrow KP_l^{00}\ (y_0, y_1, y_2)$$
$$(0,1): (y_0, y_1, y_2) \rightarrow KP_l^{01}\ (y_1, y_0, y_2)$$

In the above relations, if y_0 is equal to y_1, each permutation outputs the same value and vice versa. That is, the following equation holds:

$$y_0 = y_1 \Leftrightarrow KP_l^{00}(y_0, y_1, y_2) = KP_l^{01}(y_0, y_1, y_2)$$

Properties of KP:-

Consider: $y_0 = y_1 \Leftrightarrow KP_l^{00}(y_0, y_1, y_2) = KP_l^{01}(y_0, y_1, y_2)$
Consider: $y_1 = y_2 \Leftrightarrow KP_l^{00}(y_0, y_1, y_2) = KP_l^{10}(y_0, y_1, y_2)$
Consider: $y_0 = y_2 \Leftrightarrow KP_l^{00}(y_0, y_1, y_2) = KP_l^{11}(y_0, y_1, y_2)$
Consider: $y_0 = y_1 = y_2 \Leftrightarrow KP_l^{01}(y_0, y_1, y_2) = KP_l^{10}(y_0, y_1, y_2)$
Consider: $y_0 = y_1 = y_2 \Leftrightarrow KP_l^{01}(y_0, y_1, y_2) = KP_l^{11}(y_0, y_1, y_2)$
Consider: $y_0 = y_1 = y_2 \Leftrightarrow KP_l^{10}(y_0, y_1, y_2) = KP_l^{11}(y_0, y_1, y_2)$

2.3.2 PRINTcipher-48 Related-Key Differential Characteristics

By evaluating similar key-pairs, we shall apply a few properties to the proposed attack.:

$$(K = (SK^1, SK^2, K^* = (SK^{1*}, SK^{2*})$$
$$l = 0, ..., 15$$

Case 1 (l) $SK^1 = SK^{1*}$;
$SK_l^2 = (0,0), SK_l^{2*} = (0,1)$;
$SK_i^2 = SK_i^{2*}$where i != l;

Case 2 (l) $SK^1 = SK^{1*}$;
$SK_l^2 = (0,0), SK_l^{2*} = (1,0)$;
$SK_i^2 = SK_i^{2*}$where i != l;

Case 3 (l) $SK^1 = SK^{1*}$;
$SK_l^2 = (0,0), SK_l^{2*} = (1,1)$;
$SK_i^2 = SK_i^{2*}$where i != l;

Suppose input difference of the target round is zero. If a paired key (k, k*) satisfies Case 1(0), key-dependent permutation incorporates a nonzero-related key difference. We will construct 1-round related-key differential characteristic 0 → Case1(0) with a probability of 2^{-1} under case 1. t-round related-key differential characteristic result further can be easily extended since PRINTcipher uses the identical round key for all round.

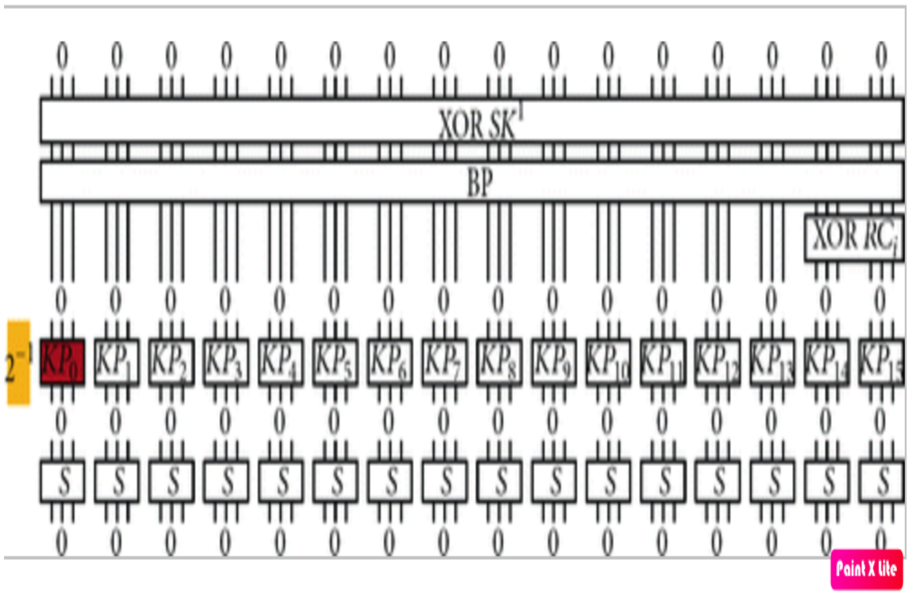


Figure 1: One-round differential characteristic of related-key under Case-1-(0)

2.3.3 PRINTcipher-48 Related-Key Cryptanalysis

On PRINTcipher-48, a differential t-round related-key features with the probability of 2^{-t} can be built. Differential characteristics of related-key rely upon concrete key value. ($K_0^{0,0}$, $K_0^{0,1}$) and ($K_0^{1,0}$, $K_0^{(1,1)}$) key pairs are used to solve this issue.

2.3.4 Basic Related-Key Attack on PRINTcipher-48

Steps for attack procedure:-

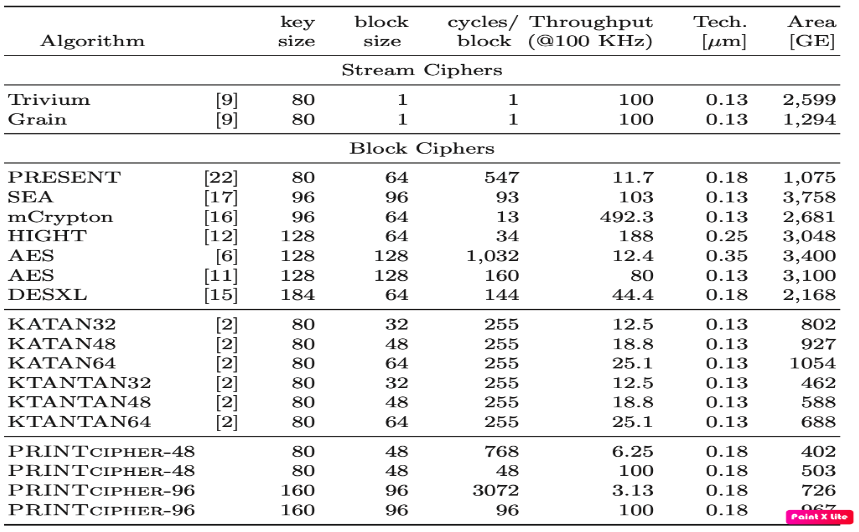
- Consider plain text strucutres of 4 plaintext each
- Wrong cipher text pair are deleted from the difference between ciphertexts.
- Output difference of round 45 should be 0 for a good ciphertext pair.
- Next, partial secret key is guessed.

2.3.5 Complexities of Basic Related-Key Attack on PRINTcipher-48

For plaintext strucutre step computational complexity is 2^{48} PRINTcipher-48 encryptions. Next, we discard wrong pair with ciphertext pairs servied is $2^{10}(= 8.2^{44}.2^{-37})$. Total computational complexity of the attack is 2^{63}.

2.4 Experiment Results

They needed to implement both PRINTcipher variations in VHDL and synthesise them using Synopsys DesignVision 2007.12 and the Virtual Silicon (VST) main cell library UMCL18G212T3, which is based on the UMC L180 0.18μm in order to calculate the effectiveness of the cipher. 1P6M logic process and has a typical voltage of 1.8 Volt



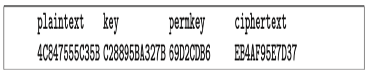
Algorithm		key size	block size	cycles/ block	Throughput (@100 KHz)	Tech. [μm]	Area [GE]
				Stream Ciphers			
Trivium	[9]	80	1	1	100	0.13	2,599
Grain	[9]	80	1	1	100	0.13	1,294
				Block Ciphers			
PRESENT	[22]	80	64	547	11.7	0.18	1,075
SEA	[17]	96	96	93	103	0.13	3,758
mCrypton	[16]	96	64	13	492.3	0.13	2,681
HIGHT	[12]	128	64	34	188	0.25	3,048
AES	[6]	128	128	1,032	12.4	0.35	3,400
AES	[11]	128	128	160	80	0.13	3,100
DESXL	[15]	184	64	144	44.4	0.18	2,168
KATAN32	[2]	80	32	255	12.5	0.13	802
KATAN48	[2]	80	48	255	18.8	0.13	927
KATAN64	[2]	80	64	255	25.1	0.13	1054
KTANTAN32	[2]	80	32	255	12.5	0.13	462
KTANTAN48	[2]	80	48	255	18.8	0.13	588
KTANTAN64	[2]	80	64	255	25.1	0.13	688
PRINTcipher-48		80	48	768	6.25	0.18	402
PRINTcipher-48		80	48	48	100	0.18	503
PRINTcipher-96		160	96	3072	3.13	0.18	726
PRINTcipher-96		160	96	96	100	0.18	967

Figure 2: comparison with some symmetric encryption algorithms.

2.5 Conclusions

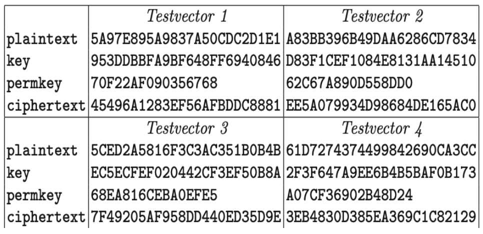
In PRINTcipher they have considered the technology of IC-printing to see how it might influence the cryptography that we use. They have proposed lightweight block cipher PRINTcipher that explicitly takes advantage of this new manufacturing approach. We related-key cryptanalysis of PRINTcipher. To recover the 80-bit secret key of PRINTcipher-48,related-key differential attack require 2^{47} related-key chosen plaintexts with a computational complexity of $2^{60.62}$. Further improvement can be done on the basic related-key attack on the full PRINTcipher-48 by considering 43-round instead of 48 round related-key.

2.6 Testvectors



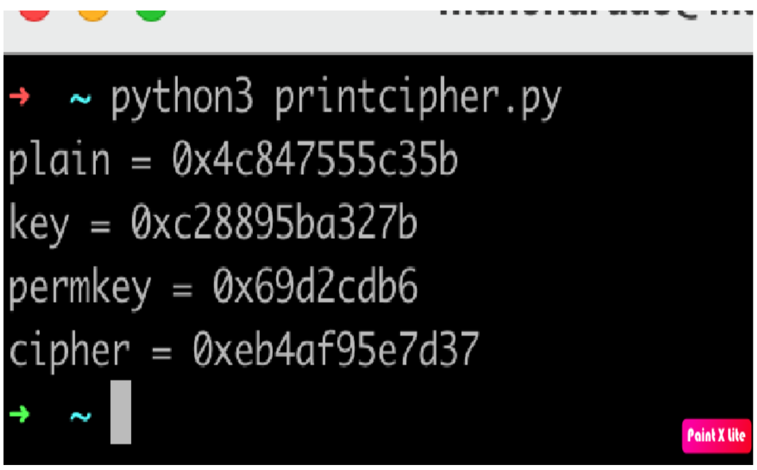
plaintext	key	permkey	ciphertext
4C847555C35B	C28895BA327B	69D2CDB6	EB4AF95E7D37

Figure 3: Sample Input and Output for PRINTcipher-48



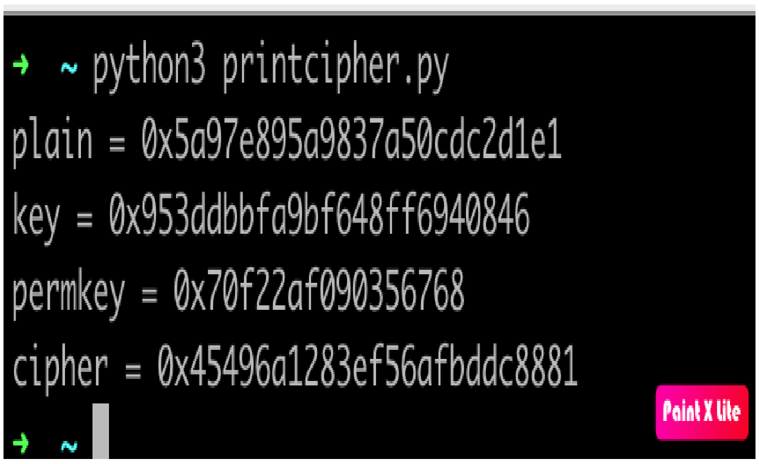
	Testvector 1	*Testvector 2*
plaintext	5A97E895A9837A50CDC2D1E1	A83BB396B49DAA6286CD7834
key	953DDBBFA9BF648FF6940846	D83F1CEF1084E8131AA14510
permkey	70F22AF090356768	62C67A890D558DD0
ciphertext	45496A1283EF56AFBDDC8881	EE5A079934D98684DE165AC0
	Testvector 3	*Testvector 4*
plaintext	5CED2A5816F3C3AC351B0B4B	61D727437449984269OCA3CC
key	EC5ECFEF020442CF3EF50B8A	2F3F647A9EE6B4B5BAF0B173
permkey	68EA816CEBA0EFE5	A07CF36902B48D24
ciphertext	7F49205AF958DD440ED35D9E	3EB4830D385EA369C1C82129

Figure 4: Sample Testvectors for the cipher



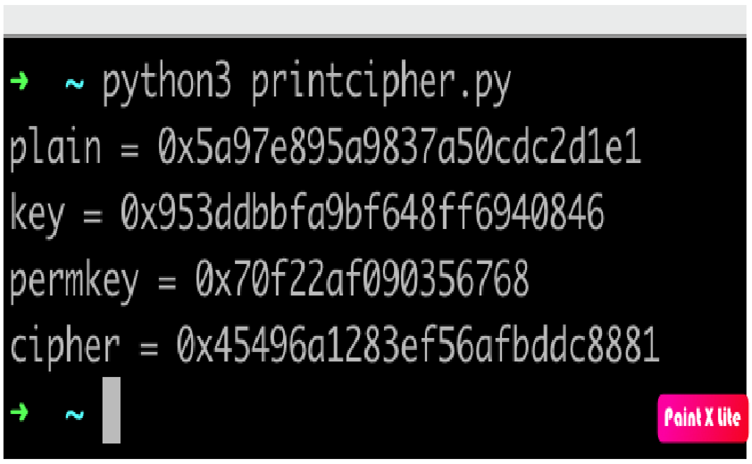
```
→  ~ python3 printcipher.py
plain = 0x4c847555c35b
key = 0xc28895ba327b
permkey = 0x69d2cdb6
cipher = 0xeb4af95e7d37
→  ~ 
```

Figure 5: Sage implementation for Encryption of above plaintext



```
➜  ~ python3 printcipher.py
plain = 0x5a97e895a9837a50cdc2d1e1
key = 0x953ddbbfa9bf648ff6940846
permkey = 0x70f22af090356768
cipher = 0x45496a1283ef56afbddc8881
➜  ~
```

Figure 6: Sage implementation for Encryption of above plaintext

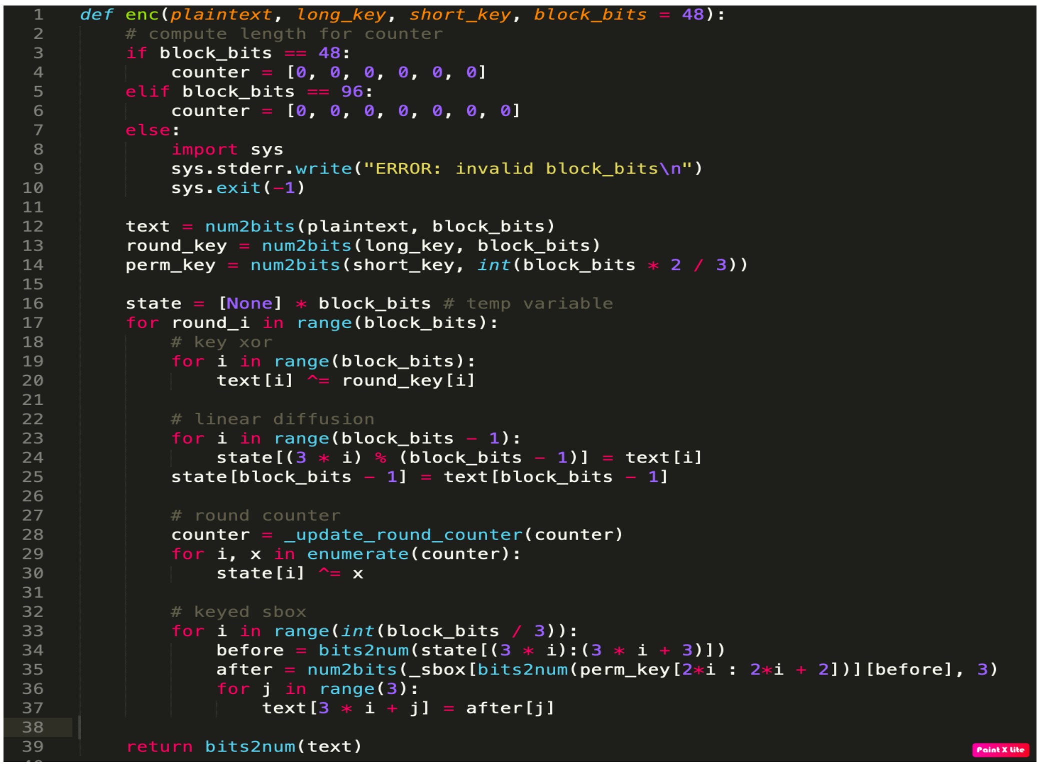


```
➜  ~ python3 printcipher.py
plain = 0x5a97e895a9837a50cdc2d1e1
key = 0x953ddbbfa9bf648ff6940846
permkey = 0x70f22af090356768
cipher = 0x45496a1283ef56afbddc8881
➜  ~
```

Figure 7: Sage implementation for Encryption of above plaintext

3 Code Snippet

The code is implemented in Python.The code snippet is given below :-



```python
def enc(plaintext, long_key, short_key, block_bits = 48):
    # compute length for counter
    if block_bits == 48:
        counter = [0, 0, 0, 0, 0, 0]
    elif block_bits == 96:
        counter = [0, 0, 0, 0, 0, 0, 0]
    else:
        import sys
        sys.stderr.write("ERROR: invalid block_bits\n")
        sys.exit(-1)

    text = num2bits(plaintext, block_bits)
    round_key = num2bits(long_key, block_bits)
    perm_key = num2bits(short_key, int(block_bits * 2 / 3))

    state = [None] * block_bits # temp variable
    for round_i in range(block_bits):
        # key xor
        for i in range(block_bits):
            text[i] ^= round_key[i]

        # linear diffusion
        for i in range(block_bits - 1):
            state[(3 * i) % (block_bits - 1)] = text[i]
        state[block_bits - 1] = text[block_bits - 1]

        # round counter
        counter = _update_round_counter(counter)
        for i, x in enumerate(counter):
            state[i] ^= x

        # keyed sbox
        for i in range(int(block_bits / 3)):
            before = bits2num(state[(3 * i):(3 * i + 3)])
            after = num2bits(_sbox[bits2num(perm_key[2*i : 2*i + 2])][before], 3)
            for j in range(3):
                text[3 * i + j] = after[j]

    return bits2num(text)
```

Figure 8: Encryption function of PRINTcipher

3.1 Software Application Implementation

The motivation behind the development of the PRINT Cipher is the issue that in RFID applications, it is not required to change the key. It is rather considered as an overhead in RFID applications. Hence, PRINT Cipher is developed which requires a single key and it has been declared efficient for many applications such as fabrication of cheap RFID tags.The other concern was using a simple key schedule where subkeys are created in a simple fashion. So this cipher fulfilled the need and no working memory is needed for the subkey computations.

Here we are implementing an NFC/RFID compatible, platform independent login tool where the credentials are secured using PRINT Cipher. The usage of PRINT Cipher here provides fast access to RFID tags and NFC devices. The application provides a secure form of authentication that has been used in smart cards and mobile authentication. The basic idea is, that once you tap an authorized tag on the reader, the appropriate password is decrypted and typed on an emulated keyboard. Once configured, you only need the HID driver on the host system. The basic idea is that when you tap a tag allowed by the reader, the corresponding password will be decrypted and entered into the emulated keyboard. Here a simulator has been developed where a login form will be shown and the credentials entered will be encrypted and secured using PRINT Cipher and when they match a secured login is successfully done. Here to demonstrate the efficiency, PRINT Cipher-48 is used in the login tool. As the cipher was designed by focusing more on the hardware efficiency, the performance of PRINT Cipher for any software implementation is slower [inefficient as compared to well known and widely used cipher like AES]than that of many well known and widely used ciphers like AES. It is observed that over a 64-bit platform, the performance of PRINT Cipher is 5-10 times slower than AES implementation.

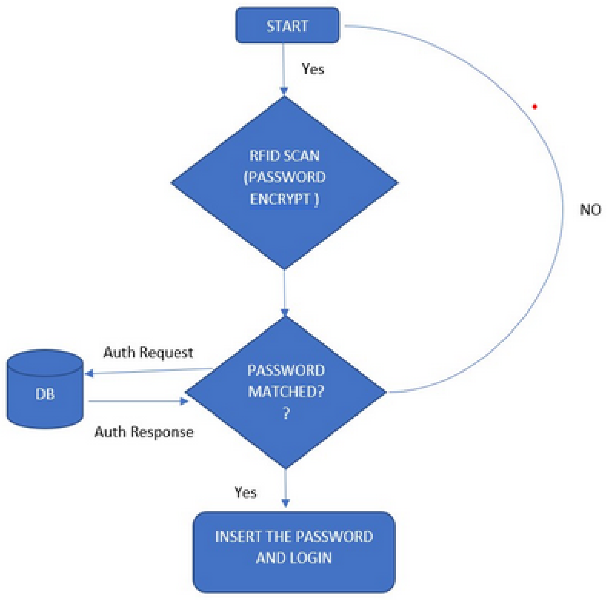


Figure 9: Figure: Login Mechanism Flow Chart

4 References

[1] Lars Knudsen, Gregor Leander, Axel Poschmann, and Matthew J.B. Robshaw, “PRINTcipher: A Block Cipher for IC-Printing”, CHES, 2010.

[2] Leander, Gregor and Abdelraheem, Mohamed Ahmed and AlKhzaimi, Hoda and Zenner, Erik, “A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack”, Springer Berlin Heidelberg, 2011.

[3] Yuseop Lee, Kitae Jeong, Changhoon Lee, Jaechul Sung, Seokhie hong, “Related-key cryptanalysis on the full PRINTcipher suitable for IC-printing“, International Journal of Distributed Sensor Networks, 2014.

[4] Tadashi Okabe, “Efficient FPGA Implementations of PRINTCIPHER”, eISSN: 2349-5162, 2016.

[5] Daemen, Joan and Rijmen, Vincent, “The Block Cipher Rijndael”, Springer Berlin Heidelberg, 2000.