Term Paper by Team Illuminati
Prince Cipher

Chodipilli Yadava Kishore[1] (11940310)
Mulukala Vivek[2] (11940720)
Srilekha Kadambala[3] (11941190)

[1] Indian Institute of Technology Bhilai , chodipilliy@iitbhilai.ac.in
[2] Indian Institute of Technology Bhilai, mulukalav@iitbhilai.ac.in
[3] Indian Institute of Technology Bhilai, srilekhak@iitbhilai.ac.in

Abstract. In this paper, we have given a detailed description of PRINCE Cipher. We have drawn the motivation after the PRINCE Cipher and also given the detailed implementation of PRINCE Cipher. We have also digged into some of the theoretical and practical attacks which can be implemented on PRINCE. This cipher is created in a way that the overhead for decryption on the top of encryption is minute and negligible. Indeed, it holds that decryption for one key parallels to encryption with a related key. This property, we refer to as α-reflection is of independent interest and we demonstrate its robustness against generic attacks.

Keywords: PRINCE · α-reflection · Differential · Integral · · FX Construction

1 Introduction

Lately, light weight cryptography has come into limelight. Especially, Algorithms of Light weight cryptography have been optimised relating to chip are but such a criteria is more particular to application. This specific method of optimisation is undeniably sound in particular cases where there are strict cost and power restrictions (for instance, passive RFID tags). Nonetheless, many other applications need instant response time and low-latency encryption. Many strong ciphers are incompatible for low-latency cryptography because execution of these ciphers take many more clock cycles. A block cipher is the only option as it takes only a single clock cycle to compute the ciphertext. But this comes at a cost of a reduced clock frequency and prolonged critical path reducing response time.. This is the motivation behind Prince Cipher, which aims to solve the problem with an implementation that can be implemented in just a few hundred thousand clock cycles.

2 Contribution of Prince Cipher:

These features are successfully achieved by Prince Cipher while implemented in hardware:

1. Generates ciphertext in no more than one clock cycle (encrypts instantaneously)
2. While implementing in the modern chip technology, lower delays were resulting in reasonably high clock cycles were accomplishes
3. Decent hardware costs (comparatively lesser than PRESENT AND AES unrolled versions)

3 Brief Description of Prince Cipher

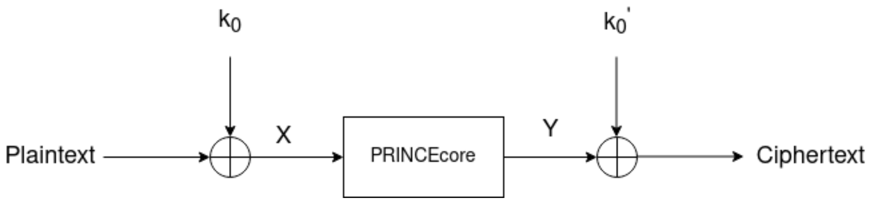
Prince Cipher operates on 64-bit blocks with 128-bit key which composes of two elements k_0 and k_1. In Prince, data is treated as a 1D array.

Prince Cipher is based on FX Construction.

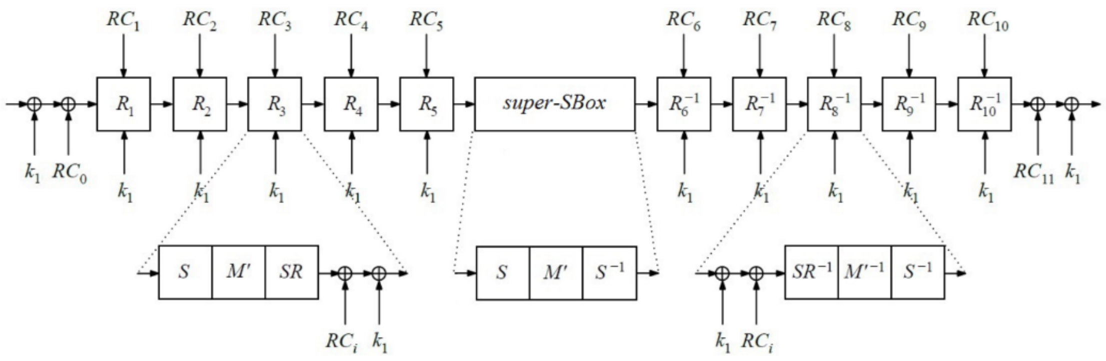
w_{in} and w_{out} are the whitening keys in the FX construction where $w_{in} = (k_0 \oplus k_1)$ and $w_{out} = k_0' \oplus k_1$. These whitening keys are XORed to the input and output of PRINCEcore respectively, specified by k_1 only.

k_0 is given by

$$k_0' = (k_0 >>> 1) \oplus (k_0 >> 63)$$



PRINCEcore: It is the 12-round core cipher.



Each round is comprised by a key addition, an non-linear layer of S-box (known as S-Layer), a linear layer known as Matrices Layer and the addition of a round constant.

1. **Key addition (k$_i$):** Each round function utilizes basic key addition (xor) with the 64-bit subkey
2. **Non-Linear Layer:** S-Layer - Uses the 4-bit S-box shown below:

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[x]	b	f	3	2	a	c	9	1	6	7	8	0	e	5	d	4

3. **Matrices Layer** - allows the property of diffusion by multiplying the state matrix with a 64x64 matrix.
4. **Round Constant Addition (RC_i):** A round constant of 64-bit is XORed with the state matrix. We define the constants given in the table below:

RC_0	0000000000000000
RC_1	13198a2e03707344
RC_2	a4093822299f31d0
RC_3	082efa98ec4e6c89
RC_4	452821e638d01377
RC_5	be5466cf34e90c6c
RC_6	7ef84f78fd955cb1
RC_7	85840851f1ac43aa
RC_8	c882d32f25323c54
RC_9	64a51195e0e3610d
RC_{10}	d3b5a399ca0c2399
RC_{11}	c0ac29b7c97c50dd

4 Prince Cipher Implementation:

4.1 Selection of S-box:

To eliminate te ability to express the cipher as a system of linear equations, the S-Layer provides non-linearity. Choosing the S-box according to the area and critical path is important because it leads to decrease the cost of encryption. However, while selecting the S-box, we should make sure it ensures the below mentioned criteria to ensure the security of the decryption keys.

- Maximal differential probability is 0.25
- 15 differentials exist with 0.25 probability
- Maximal linear approximation absolute bias is 0.25
- 30 linear approximations exist with an absolute bias value of 0.25
- Each of the 15 non-zero component function has algebric degree 3.

8 S-boxes exist which satisfy the above conditions.

4.2 α Reflection Property:

Symmetric property is one of the fascinating aspects of PRINCE cipher. Round constants satisfy the condition -> for i $\in [0, 11]$, $RC_i \oplus RC_{11-i} = \alpha$, where α is a constant. In this case, value of α is c0ac29b7c97c50dd. RC_0 is 0, RC_j where $1 \leq j \leq 5$ are acquired from fractional part of π.

We can get the inverse of PRINCEcore from this α value. Since M' is an involution, inverse of PRINCEcore specified by k is equivalent to PRINCEcore specified by $k \oplus \alpha$. This condition is known as α reflection property.

For a key $(k_0||k_0'||k_1)$,

$$D_{(k_0||k_0'||k_1)}(.) = E_{(k_0'||k_0||k_1 \oplus \alpha)}(.)$$

Therefore, one has to do a minimal change for decryption, then utilize the same as that from encryption.

4.3 The Linear Layer

It comprises of 2 operations:

1. Shift Rows

2. M' Layer

Both the operations are performed in continuity, however, M' layer is used individually. On a consequence of tis, M' layer must be an INVOLUTION to retain the cipher's symmetry. On the flip side, since the M layer exists in the forward rounds and backward rounds in an exact opposite order to reverse the effect while deciphering, it need not be an involution. Since our goal is to acomplish diffusion, we use shift rows along with M' layer in backward and forward rounds.

Shift Rows operation in Prince Cipher acts in the same way as in Shift rows of AES and it rearranges the nibbles in the given way:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

The Matrix taken in the M' layer must have as lesser amount of 1's as possible to ease the computation. It helps in reducing the implementation cost. Simultaneously, for diffusion and security purposes, there should be minimum sixteen active S-Boxes in four consecutive rounds. Therefore, number of 1's per row and per column has been taken as three to maintain the stability between the cost and security.

$$M_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The above 4x4 matrices are used as frameworks for M' (64x64).

Next, we prepare two matrices $M^{(0)}$ and $M^{(1)}$ which are of dimension 64x64:

$$\vec{M^{(0)}} = \begin{bmatrix} M_0 & M_1 & M_2 & M_3 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{bmatrix} \quad \vec{M^{(1)}} = \begin{bmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{bmatrix}$$

These matrices are constructed such that each column and row is a chosen permutation of $M_1, M_2, M3$ and M_4 matrices. But the constructed $M^{(0)}$ and $M^{(1)}$ should also involutions i.e, they are inverses of themselves.

$$\begin{pmatrix} M_0 & 0 & 0 & 0 \\ 0 & M_1 & 0 & 0 \\ 0 & 0 & M_1 & 0 \\ 0 & 0 & 0 & M_0 \end{pmatrix}$$

Then we utilize the 16x16 matrices which are obtained above along the diagonal of M' layer. Remaining all are 0s. With tis, M' is an incolution with 2^32 fixed points, which implies to the states which left unaffected after the M' layer operates on those states.

4.4 Key Expansion:

In PRINCE cipher, the 128-bit key $(k_0||k_1)$ is expanded (rooted on FX Construction) to get a 192 bit key $(k_0||k_0'||k_1)$ by mapping

$$(k_0||k_1) \rightarrow (k_0||P(k_0)||k_1)$$

Key expansion is based on FX Construction. Since k_1 is utilized in 12 rounds of prince cipher, there is a chance of a cryptanalyst partially guessing k_1. To block this, key explansion uses FX construction.
The set of all triples $(k_0||P(k_0)||k_1)$ should correspond to an MDS code of length 3 and size 2^{128} over F_2^{64} i.e, both $x \rightarrow P(x)$ and $x \rightarrow x \oplus P(x)$ should be permutations of F_2^{64}.

Thus, a hardware-favaourable choice for P such that both$P$ and $P \oplus Id$ are permutations is

$$P(x) = (x >> 1) \oplus (x >> 63),$$

Now, we can confirm that $P(x) = 0$ (respectively, $P(x) = x$) has a distinct solution.

5 Practical Attacks

5.1 Integral Cryptanalysis

5.1.1 4-Round Integral Attack

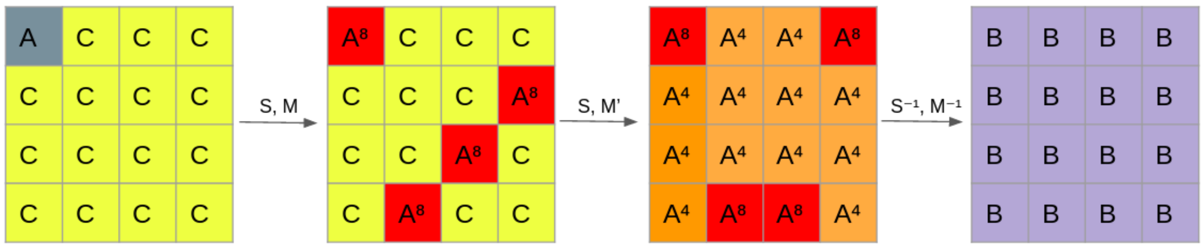
We are performing the integral attack on round-reduced PRINCE.

- **FOR EVEN ROUNDS:** Added same no. of rounds before and after the middle rounds
- **FOR ODD ROUNDS:** Added one extra round at the start.

We start with one out of sixteen active nibbles in the plaintext. Since we know that all nibbles are balanced after 3.5 rounds, we estimate the key with the help of partial decryption.
Hence we start with

- 1-nibble containing the 'ALL PROPERTY'
- remaining containing the 'CONSTANT PROPERTY'
- Ending with all balanced nibbles at the end of 3.5 rounds

Representation of the above description is given below: (In the figure, A^i is the nibble taking i distinct values)



- Balance property after 3.5 rounds will be destoyed because of the transformation done by S-Box. We estimate the value of $(k_0 \oplus k_1)$(let us say 'y') and undergo partial decryption with final S-Box to verify if all nibbles are balanced.
- We go with 5 sets of 16 plaintexts (2^4) each and recognize that y which is present in right key guesses for all sets
- After identifying y, we should get k_1
- Beginning with 4 active nibbles, again we go with 5 sets of 16 plaintexts (2^4) (refer to 2nd block in the above diagram)
- Now, with the identified y value, remove fourth round by performing XOR with the ciphertext.
- Next, perform the linear layer inversion & for all possible key guesses of nibbles in k_1, undergo partial decryption using 1st S-Box
- The one which provides a BALANCED NIBBLE is CORRECT NIBBLE key guess
- Then we recover k_0' using y and k_1
- We can recover k_0 from k_0' by performing a set of LINEAR OPERATIONS.

DataComplexity= $2 \times 5 \times 2^4 = 2^7$ since we have chosen five sets of 2^4 plaintexts two times in the attack above.

TimeComplexity=$16 \times 2 \times 5 \times 2^4 = 2^{11}$. Since Each alteration of S-box : one operation; One for each nibble, S-Box is performed sixteen times for each plaintext. Total Plaintexts are $2 \times 5 \times 2^4$.

5.2 Differential Attacks

In differential attacks, we study between the trail from middle to plaintext and trail from middle to ciphertext. This Version has a $n - x - n$ construction. Important points about this are given below:

1. PRE-WHITENING with k_1 and RC_0
2. n-forward rounds
3. Super SBox part $[S, M', S^{-1}]$ (middle part)

4. n-backward rounds
5. PRE-WHITENING with k_1 and RC_{11}

- There exists four nibbles (2^8) each which pass through $M^{(}0)$ and $M^{(}1)$ without change
- Hence, there are 2^{32} states which can go unaffected through M'-layer

5.2.1 SBOX ANALYSIS

DDT for S-Box of Prince Cipher:

in/out	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	No. of solutions
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	4	0	0	2	0	2	0	4	2	0	2	0	0	0	0	6
2	0	2	0	4	0	0	0	2	2	0	0	0	0	4	2	0	6
3	0	0	0	0	0	2	2	0	2	2	2	2	2	0	0	2	8
4	0	2	2	4	2	2	0	0	2	0	2	0	0	0	0	0	7
5	0	0	2	2	0	2	0	2	0	2	0	2	2	2	0	0	8
6	0	0	2	2	0	2	2	0	0	2	0	2	0	0	4	0	7
7	0	0	2	0	0	0	2	0	2	0	4	0	0	2	2	2	7
8	0	0	2	0	4	2	0	0	2	2	0	2	0	2	0	0	7
9	0	0	2	2	0	0	0	0	0	2	2	0	4	2	0	2	7
a	0	0	0	2	2	4	0	4	2	0	0	0	0	0	0	2	6
b	0	2	0	0	4	0	0	2	0	0	0	2	2	0	2	2	7
c	0	4	0	0	0	2	2	0	0	0	2	2	2	0	2	0	7
d	0	2	0	0	0	0	0	2	0	4	2	0	0	2	2	2	7
e	0	0	2	0	0	0	4	2	0	0	0	2	2	2	0	2	7
f	0	0	2	0	2	0	2	2	0	0	2	0	2	0	2	2	8

Final column depicts possible number of output differences for each of the input differences.

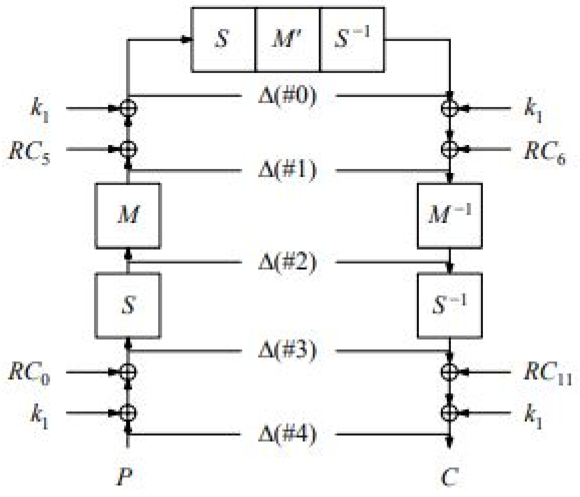
DDT for Inverse S-Box of Prince Cipher:

in/out	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	No. of solutions
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	4	2	0	2	0	0	0	0	0	0	2	4	2	0	0	6
2	0	0	0	0	2	2	2	2	2	2	0	0	0	0	2	2	8
3	0	0	4	0	4	2	2	0	0	2	2	0	0	0	0	0	6
4	0	2	0	0	2	0	0	0	4	0	2	4	0	0	0	2	6
5	0	0	0	2	2	2	2	0	2	0	4	0	2	0	0	0	7
6	0	2	0	2	0	0	2	2	0	0	0	0	2	0	4	2	7
7	0	0	2	0	0	2	0	0	0	0	4	2	0	2	2	2	7
8	0	4	2	2	2	0	0	2	2	0	2	0	0	0	0	0	7
9	0	2	0	2	0	2	2	0	2	2	0	0	0	4	0	0	7
a	0	0	0	2	2	0	0	4	0	2	0	0	2	2	0	2	7
b	0	2	0	2	0	2	2	0	2	0	0	2	2	0	2	0	8
c	0	0	0	2	0	2	0	0	0	4	0	2	2	0	2	2	7
d	0	0	4	0	0	2	0	2	2	2	0	0	0	2	2	0	7
e	0	0	2	0	0	0	4	2	0	0	0	2	2	2	0	2	7
f	0	0	0	2	0	0	0	2	0	2	2	2	0	2	2	2	8

IMPORTANT: After summing up the numbers in solutions column for each, we get 106. i.e, for 256 input pairs, 106 output differences exist. Mean non zero value in DDT = $\frac{256}{106}$ = **2.415**

5.2.2 2-Round Inside-Out Attack

In this attack, we will use these states as the initial difference beginning from the middle rounds. The differential trail is as follows:



$$\xrightarrow[p=2^{-32}]{S^{-1},M',S} 0 \xrightarrow[p=1]{\oplus k1 \oplus RC_5/RC_6} \alpha \xrightarrow[p=1]{M^{-1}} \beta \xrightarrow[p=1]{S^{-1}} \gamma \xrightarrow[p=1]{\oplus k1 \oplus RC_{11}/RC_0} \gamma \oplus \alpha$$

PROCESS OF DIFFERENCE PROPOGATION

- If we observe, the difference is among intermediate stages but not among plaintexts. The difference is in the intermediate stages which are symmetrical about (S, M', S^{-1}), and lastly among ciphertext and plaintext.
- We denote it at different stages as $\Delta(\#i)$

The propogation process is detailed below:

- **KEY ADDITION AND ROUND CONSTANT ADDITION**
  - The addition of key doesn't affect the difference since it's the same key. Difference in round constants is α, so $\Delta(\#0) = \alpha \oplus \Delta(\#1)$
- **M or M INVERSE**
  - Since it provides diffusion, changes in the difference in both sides are diffused, nonetheless, we get the effecting difference with probability 'one'.

- **<u>S-BOX</u>**
  - Provides non-linearity therefore we could only know with how much probability value does the difference propogate based on corresponding Difference Distribution Table

<u>DESCRIPTION OF ATTACK:</u>

- Choose 2^{32} plaintexts. We have $\gamma_i \oplus \alpha = P_i \oplus C_i$
- By applying M' layer to $\alpha = (c0ac||29b7||c97c||50dd)$, with '1' probability, we get $\beta = (42a3||356a||5d3a||0fe3$
- From β we get potential values of γ. This turns out to be $6 \times 8 \times ... \times 6 \approx 2^{41.38}$. Filter out plaintext and ciphertext pairs and expected value to remain is $2^{9.38}$
- For every nibble in every remaining P_i, we lookup all possible S solutions a,b,c,d $\in$ $0, 1^4$ with $a \oplus b = \gamma \xrightarrow{S} \beta = c \oplus d$
- There are $(2^{1.27} \times 16)$ $\approx 2^{20.35}$ solutions in average for every P_i for state$(\#3)_i$, which enumerate by $(\#3)_i^j$. $(\#3)_i^j = P_i + RC_0 + k_1$
- $(\#3)_i^j$ guess $(k_1)_i^j$ and verify if computed cipher is actual ciphertext C_i

  <u>DATA AND MEMORY COMPLEXITY</u>$= 2^{32}$

  <u>TIME COMPLEXITY</u> $= 2^{32} + 2^{29.73} = 2^{32}$

6 Software Application:

A web app written in python(backend code) and Javascript(frontend code).
We used two frameworks, FastAPI for backend development and NextJS(based on ReactJS) for frontend development.
We developed a simple encryption and decryption utility with prince cipher along with a chat application, where we used socket programming and prince cipher for encrypting and decrypting the messages and SHA256 for password safety, where passwords are stored in the server. Encryption and Decryption: Here we take any length message in ascii characters string and key as fixed length of 16 ascii character string. Our core prince cipher code takes message length of 16 hexadecimal values and key length of 32 hexadecimal values and does the encryption and decryption. For generalization to humans what we had done is, we take input of any length and add required padding to make the total length as a multiple of 8. As padding we will add character c(c stands for crypto) at the end of the message. Now we start taking 8 characters at once, make each character into its ascii number and convert it into an 8 bit binary number. From this 8 bit binary number we take the first 4 bits and convert it to a hexadecimal number and later we convert the next 4 bits to a hexadecimal number. We perform the same steps for each of the characters in the 8 bit chunk. Now we pass these 16 hexadecimal numbers as a string to our prince encrypter code along with the key converted to hexadecimal in the same way. This code outputs 16 hexadecimal numbers. Like this we do this for all 8 character chunks from the original message and concatenate all 16 hexadecimal output strings. This concatenated string is the end result one will see after the encryption of the whole message. Decryption also includes almost the same steps. First it takes the message which is a multiple of 16, all hexadecimal numbers (essentially decryption message is an output from the encryption) and key as a string of 16 ascii characters which are converted to 32 hexadecimal numbers during the decryption process internally in the same procedure shown above. Now the

description code takes an input of a string containing 16 hexadecimal numbers and a key which is a string of 32 hexadecimal numbers. It converts all chunks of 16 hexadecimal numbered strings to corresponding deciphered messages and concatenates them all. Now the message will contain the padded characters at the end(previously we appended some number of c letters at the end), so as a final step of decryption we delete the ending c letters from the final string and output it as the final deciphered string.

6.1 Group Chat Application

This is a simple group chat application developed using sockets(UDP). This simple chat application has features like user registration, password safety and supports any number of users and message encryption where encryption and decryption are done through prince cipher. Since our prince cipher code is written in python code and our frontend code is in Javascript, we created a mini server in the main server to mock it as client. Here we just pass the messages, keys and passwords for the backend miniserver which mocks the client and encrypts the messages and sends them to the server, where the sent messages are stored in the main server. Now to get the messages(which are from the other people and stored in the server) the user needs to request the main server for messages with the password and username. Here the passwords are not stored directly in the server. Before saving passwords are hashed and then saved. This procedure makes the passwords more safe, as the password itself is not stored and can't be accessed by the administrator or hackers. We provided the message sending interface for sending the message and receiving the messages from other users. To send the message one has to give the username and password. Also to view the messages the procedure is the same. We also included the user registration interface where one can register to the group chat. We used the same encryption and decryption strategies in the Encryption and Decryption section. We included five registered users by default in the group chat application. Three of the five default users are the Team Illuminati members and the rest two are guest accounts. A new user can either use these guest accounts or can register in the New registration section.

7 Conclusion

We have successfully implemented PRINCE CIPHER using python, the file is named as $software\_implementation$.ipynb. We have also detailed the S-Box analysis, Key Expansion and Linear Layer and S-Box analysis file is named as $sbox\_analysis$.ipynb. We have discussed in detail about Differential and Integral Cryptanalysis of PRINCE Cipher. As a part of brownie point section, we have implemented 4-round Integral Attack, file is named as $integral\_cryptanalysis$.ipynb. We have also implemented a software application 'GROUP CHAT APPLICATION' using Prince Cipher.

8 Refernces

1. On the security of the core of PRINCE against biclique and differential cryptanalysis [link]
2. Truncated differential cryptanalysis of PRINCE [link]
3. Multiple Differential Cryptanalysis of Round-Reduced PRINCE [link]
4. PRINCE – A Low-latency Block Cipher for Pervasive Computing Applications [link]
5. Practical Attacks on the Round-reduced PRINCE [link]
6. Reflection Cryptanalysis of PRINCE-like Ciphers [link]