vishwaCTF

CHALLENGE NAME : [BITBANE]

DEV : [SAKSHAM SAIPATWAR]

CATEGORY : [CRYPTO]

LEVEL : [HARD]

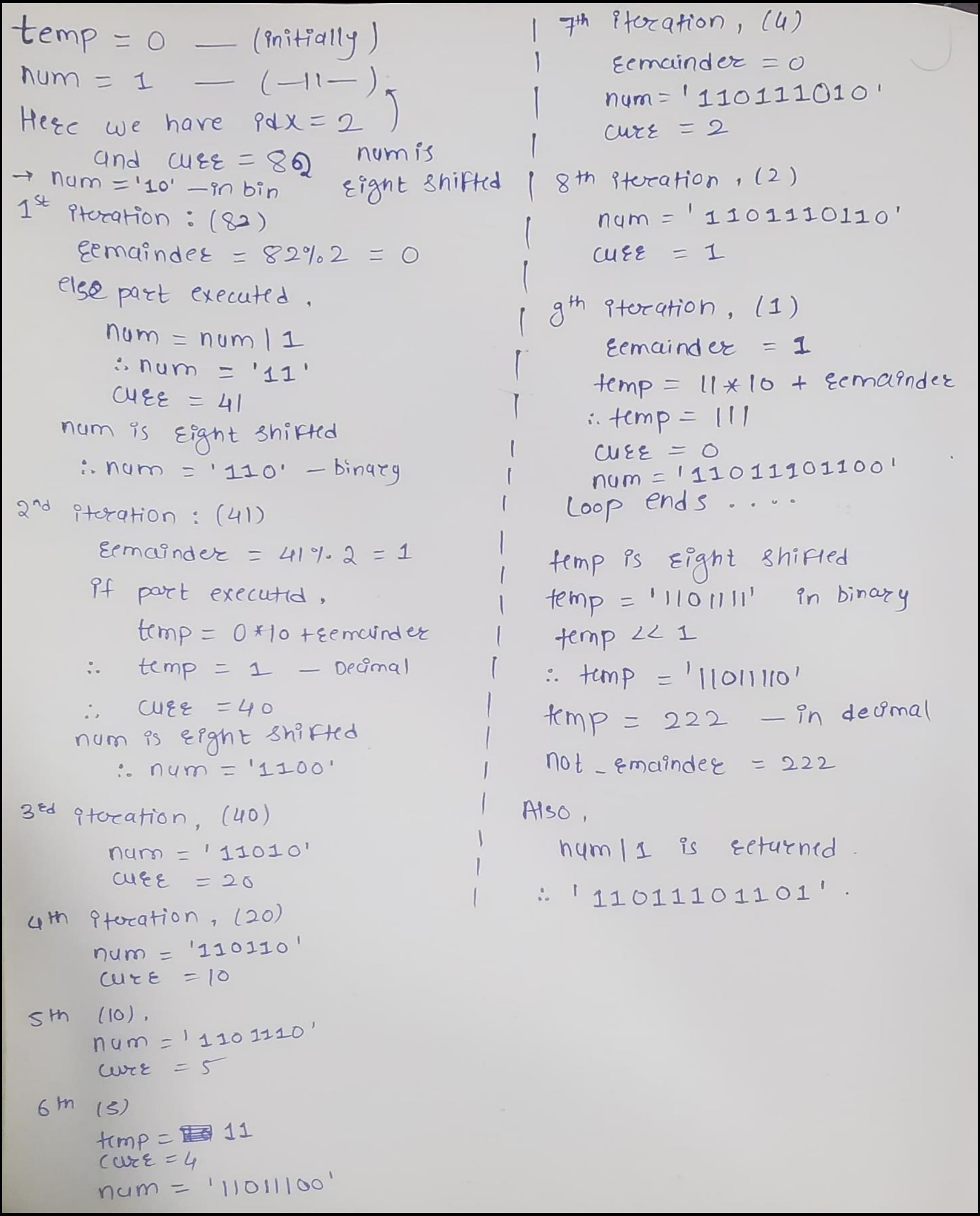
2024

This challenge is based on concept of bit manipulation (Won't be hiding anything ... it was supposed to be a tough one but due to an unintentional flaw , the flag could be brute-forced as the encryption was only dependant on the current position of the char and its ASCII value) . Anyways let me walk you through the logic used for encryption .

Suppose the initial character is 'R', and its ASCII value is '82'. According to the code written in the encode function, the value of the variable 'idx' will come out to be '(0 % 8) + 2', which evaluates to '2'. (There's a reason behind restricting the index value between 2 and 9; try if you could figure it out.) Now both this ASCII value and the 'idx' we just calculated are passed to the 'create' function, which does the real work.

The create function, as seen, is divided into three parts: one is to create the topping, the next is to create the base, and finally, to put the topping on the base. (I've taken complete care to ensure that no data is lost in this bitwise OR operation).

The createTopping function takes three parameters: the ASCII value of the current character, the calculated idx, and the reference of the variable named 'not_remainder' (which actually isn't any remainder) to store the value inside it. Inside this function, two local variables are used: 'temp' (just a dummy for 'not_remainder') and 'num', which is finally returned. (If you observe carefully, initially it's '1', and also while returning, it's ORed with '1'. This is just to enable you to differentiate between the base and the topping at the time of decryption.) The image below explains how things are happening..



temp = 0 — (initially)

num = 1 — (—11—)

Here we have idx = 2)

and curr = 82   num is right shifted

→ num = '10' — in bin

1st iteration : (82)

remainder = 82%2 = 0

else part executed.

num = num | 1

∴ num = '11'

curr = 41

num is right shifted

∴ num = '110' — binary

2nd iteration : (41)

remainder = 41%2 = 1

if part executed,

temp = 0*10 + remainder

∴ temp = 1 — Decimal

∴ curr = 40

num is right shifted

∴ num = '1100'

3rd iteration, (40)

num = '11010'

curr = 20

4th iteration, (20)

num = '110110'

curr = 10

5th (10),

num = '1101110'

curr = 5

6th (5)

temp = 11

curr = 4

num = '11011100'

7th iteration, (4)

remainder = 0

num = '110111010'

curr = 2

8th iteration, (2)

num = '1101110110'

curr = 1

9th iteration, (1)

remainder = 1

temp = 11*10 + remainder

∴ temp = 111

curr = 0

num = '11011101100'

Loop ends....

temp is right shifted

temp = '1101111' in binary

temp << 1

∴ temp = '11011110'

temp = 222 — in decimal

not_remainder = 222

Also,

num|1 is returned.

∴ '11011101101'.

(IN THE IMAGE ABOVE, I MISTAKENLY WROTE 'RIGHT SHIFTED' INSTEAD OF 'LEFT SHIFTED'. MAKE SURE YOU FIX IT WHILE READING)

Now, in the next step, the value inside '*not_remainder*' (if present) is converted into bits from behind, so it gets reversed. Then, the additional bits are appended at the end.

Finally, the topping is ORed with the base to get the encrypted number. Later, this number is passed into functions like '*applyKey* ' and '*extraSecurity* ', which can be easily reversed by applying the same functions again.

Note: In the '*checkValidity* ' function, if you think it's checking for a prime number, then you are wrong. Upon careful observation, you will find that it will return true for '*4* ', which isn't prime!

I will be providing the code to get the flag using brute force. However, I would suggest you try to solve this challenge without brute forcing—it would really test your minds. After this thorough explanation of how this encryption works, I don't think it would be difficult to code.

FLAG :-

VishwaCTF{BIT5_3NCRYPT3D_D3CRYPTED_M1ND5_D33PLY_TE5T3D}

```
int main()
{
   fstream file;
   file.open("Encrypted.txt");
   if (!file)
   {
      cout << "Error opening file!";
      return 1;
   }

   vector<int> encryptedText;
   int number;
   while (file >> number)
   {
      encryptedText.push_back(number);
   }
   file.close();
   string key = "VishwaCTF";
   extraSecurity(encryptedText);
   applyKey(encryptedText, key);
   string temp =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ{}_";
   int n = temp.size();
   int len = encryptedText.size();
   for (int i = 0; i < len; ++i)
   {
      int idx = (i % 8) + 2;
      for (int j = 0; j < n; ++j)
      {
         int curr = temp[j];
         int num = create(curr, idx);
         if (num == encryptedText[i])
         {
            cout << temp[j];
            break;
         }
      }
   }
   return 0;
}
```

DEV : [SAKSHAM SAIPATWAR]