

The cipher used remains a mystery, requiring your keen reverse engineering skills. Only those who delve deep, find decryption cunningly, and think beyond the conventional can unravel the secrets within. Approach with curiosity, analyze meticulously, and piece together the puzzle. Are you up for the challenge? Dive in, crack the code, and claim your victory!

### Analysing the File:

```
(kali㉿kali)-[~/Documents]
$ file rev.exe
rev.exe: PE32+ executable (console) x86-64, for MS Windows, 15 sections
```

PE32+ is the Portable Executable 64-bit format. PE files are the standard file format for executables, object code, and Dynamic Link Libraries (DLLs) in Windows operating systems.

### Strings:

```
$ strings rev.exe | grep NOVA{
NOVA{Fake_flag}
NOVA{hunt_for_fake_flag}
NOVA{Search_for_decryption}
NOVA{This_is_not_the_flag}
NOVA{FLAG_NOT_FOUND}
NOVA{TRY_HARDER_NEXT_TIME}
NOVA{FLAG_IS_IN_ANOTHER_CASTLE}
NOVA{FLAG_MAYBE_IN_THE_CLOUDS}
NOVA{FLAG_NOT_YET_IMPLEMENTED}
NOVA{FLAG_IS_BEYOND_YOUR_REACH}
NOVA{FLAG_LOOKS LIKE_A_UNICORN}
```

It seems everything is fake flags.

To gather some context, run this in Windows. You're prompted to enter

Enter Encrypted string Without Space and Key. By entering the random string and key we will get the Decrypted text.

```
G:\>rev.exe
Welcome to NOVA-CTF2024
-----
Enter Encrypted string Without Space:tfahjdklsfdsa
Enter the key: dfghjklasdfads
Decrypted text: ãOP\r
```

As this is a reversing challenge static analysis would be helpful. By using the decryption process We need to find the cipher and look for the encrypted string and key to get the flag.

The decryption algorithm used in this challenge is a variant of the RC4 stream cipher. It takes the ciphertext and key as inputs and returns the decrypted plaintext. The algorithm initializes an array `s` with values from 0 to 255, shuffles the values based on the key, and then uses the shuffled array to decrypt the ciphertext.

After opening in IDA

## Looking into Main Function

```
call puts
lea rcx, asc_405148 ; "-----"
call puts
lea rcx, Format ; "Enter Encrypted string Without Space:"
call printf
mov ecx, 0 ; Ix
mov rax, cs:__imp__acrt_iob_func
call rax ; __acrt_iob_func
mov rdx, rax
lea rax, [rbp+4A0h+hex_string]
mov r8, rdx ; Stream
mov edx, 3E8h ; MaxCount
mov rcx, rax ; Buffer
call fgets
lea rax, [rbp+4A0h+hex_string]
lea rdx, Control ; "\n"
mov rcx, rax ; Str
call strcspn
mov [rbp+rax+4A0h+hex_string], 0
lea rax, [rbp+4A0h+hex_string]
mov rcx, rax ; Str
call strlen
mov [rbp+4A0h+hex_len], eax
mov eax, [rbp+4A0h+hex_len]
mov edx, eax
shr edx, 1Fh
add eax, edx
sar eax, 1
movsxd rdx, eax
sub rdx, 1
mov [rbp+4A0h+var_50], rdx
movsxd rdx, eax
mov [rbp+4A0h+var_4E0], rdx
mov [rbp+4A0h+var_4D8], 0
movsxd rdx, eax
mov [rbp+4A0h+var_4F0], rdx
mov [rbp+4A0h+var_4E8], 0
cdqe
add rax, 0Fh
shr rax, 4
shl rax, 4
call __chkstk_ms
sub rsp, rax
lea rax, [rsp+520h+var_4F0]
add rax, 0
mov [rbp+4A0h+p_ciphertext], rax
mov [rbp+4A0h+i], 0
jmp short loc_401DC6
```

```

loc_401DC6:
mov     eax, [rbp+4A0h+i]
cmp     eax, [rbp+4A0h+hex_len]
jl      short loc_401D82

```

```

loc_401D82:
mov     eax, [rbp+4A0h+i]
mov     edx, eax
shr     edx, 1Fh
add     eax, edx
sar     eax, 1
cdqe
mov     rdx, [rbp+4A0h+p_ciphertext]
lea     rcx, [rax+rdx]
mov     eax, [rbp+4A0h+i]
cdqe
lea     rdx, [rbp+4A0h+hex_string]
add     rax, rdx
mov     r8, rcx
lea     rdx, a2hhx      ; "%2hhX"
mov     rcx, rax        ; Buffer
call    sscanf
add     [rbp+4A0h+i], 2

```

```

mov     eax, [rbp+4A0h+hex_len]
mov     edx, eax
shr     edx, 1Fh
add     eax, edx
sar     eax, 1
mov     [rbp+4A0h+ciphertext_length], eax
lea     rcx, aEnterTheKey ; "Enter the key: "
call    printf
mov     ecx, 0          ; Ix
mov     rax, cs:__imp__acrt_iob_func
call    rax ; __acrt_iob_func
mov     rdx, rax
lea     rax, [rbp+4A0h+key]
mov     r8, rdx          ; Stream
mov     edx, 64h ; 'd' ; MaxCount
mov     rcx, rax          ; Buffer
call    fgets
lea     rax, [rbp+4A0h+key]
lea     rdx, Control    ; "\n"
mov     rcx, rax          ; Str
call    strcpy
mov     [rbp+rax+4A0h+key], 0
mov     eax, [rbp+4A0h+ciphertext_length]
add     eax, 1
mov     rdx, rsp
mov     rdi, rdx
movsxd  rdx, eax
sub     rdx, 1
mov     [rbp+4A0h+var_68], rdx
movsxd  rdx, eax
mov     r14, rdx
mov     r15d, 0
movsxd  rdx, eax
mov     r12, rdx
mov     r13d, 0
cdqe
add     rax, 0Fh
shr     rax, 4
shl     rax, 4

```

```

movsxd rdx, eax
mov     r12, rdx
mov     r13d, 0
cdqe
add     rax, 0Fh
shr     rax, 4
shl     rax, 4
call    __chkstk_ms
sub     rsp, rax
lea     rax, [rsp+520h+var_4F0]
add     rax, 0
mov     [rbp+4A0h+p_decrypted_string], rax
mov     rbx, [rbp+4A0h+p_decrypted_string]
lea     rax, [rbp+4A0h+key]
mov     rcx, rax ; Str
call    strlen
mov     r8d, eax
mov     rax, [rbp+4A0h+p_ciphertext]
lea     rcx, [rbp+4A0h+key]
mov     edx, [rbp+4A0h+ciphertext_length] ; ciphertext_length
mov     [rsp+520h+plaintext], rbx ; plaintext
mov     r9d, r8d ; key_length
mov     r8, rcx ; key
mov     rcx, rax ; ciphertext
call    decrypt
mov     rdx, [rbp+4A0h+p_decrypted_string]
mov     eax, [rbp+4A0h+ciphertext_length]
cdqe
mov     byte ptr [rdx+rax], 0
mov     rax, [rbp+4A0h+p_decrypted_string]
mov     rdx, rax
lea     rcx, aDecryptedTextS ; "Decrypted text: %s\n"
call    printf
mov     eax, 0
mov     rsp, rdi
mov     rsp, rsi
lea     rsp, [rbp+468h]
pop     rbx
pop     rsi
pop     rdi
pop     r12
pop     r13
pop     r14
pop     r15
pop     rbp
retn
main endp

```

opening the decrypt function,



```

idiv    [rbp+0A0h+key_length] ; Divide EDX:EAX by key_length
mov     eax, edx               ; Store the remainder (modulus) in EAX
cdqe                                ; Sign-extend EAX to RAX
mov     rdx, [rbp+0A0h+key]    ; Load the address of key into RDX
add     rax, rdx               ; Add the offset to the base address of key
movzx   eax, byte ptr [rax]    ; Load the byte value from the key into EAX
movzx   eax, al                ; Zero-extend AL to AX

```

These instructions calculate the key index based on the current `i` index, retrieve the corresponding byte from the key, and store it in EAX.

#### Update `j` Index:

```

lea     edx, [rcx+rax]         ; Calculate the sum of the previously
                                calculated index and the key value
sar     eax, 1Fh               ; Arithmetic shift right to get the sign bit
                                into all bits
shr     eax, 18h               ; Shift right by 24 bits to extract the lowest
                                byte
add     edx, eax               ; Add the extracted byte to the sum
movzx   edx, dl                ; Zero-extend DL to EDX
sub     edx, eax               ; Subtract the sign-extended key byte from EDX
mov     eax, edx               ; Move the updated value of j into EAX
mov     [rbp+0A0h+j], eax      ; Store the updated value of j

```

These instructions update the value of the `j` index based on the calculated sum and the extracted byte from the key.

#### Update `s[i]` and `s[j]`:

```

mov     eax, [rbp+0A0h+i]      ; Load the value of `i` into EAX
cdqe                                ; Sign-extend EAX to RAX
movzx   eax, [rbp+rax+0A0h+S]  ; Load the value of `S[i]` into EAX
mov     [rbp+0A0h+temp], al     ; Store the value of `S[i]` in a temporary
                                variable
mov     eax, [rbp+0A0h+j]      ; Load the value of `j` into EAX
cdqe                                ; Sign-extend EAX to RAX
movzx   eax, [rbp+0A0h+temp]   ; Load the temporary value into EAX
mov     edx, [rbp+0A0h+i]      ; Load the value of `i` into EDX
movsxd  rdx, edx               ; Sign-extend EDX to RDX

```

```
mov     [rbp+rdx+0A0h+S], al      ; Store the value of `S[i]` in memory at the
correct index
```

These instructions swap the values of `S[i]` and `S[j]` by temporarily storing `S[i]`, retrieving `S[j]`, and then updating `S[i]` and `S[j]` in memory.

**Increment `i` Index:**

```
add     [rbp+0A0h+i], 1          ; Increment the value of i by 1
```

This instruction increments the value of the `i` index, preparing for the next iteration of the decryption loop.

```
mov     eax, eax
sar     eax, 1Fh
shr     eax, 18h
add     edx, eax
movzx   edx, dl
sub     edx, eax
mov     eax, edx
mov     [rbp+0A0h+j], eax
mov     eax, [rbp+0A0h+i]
cdq     [rbp+rax+0A0h+S]
movzx   eax, [rbp+0A0h+temp_0], al
mov     eax, [rbp+0A0h+j]
cdq     [rbp+rax+0A0h+S]
movzx   eax, [rbp+0A0h+i]
mov     edx, [rbp+0A0h+i]
movsxd  rdx, edx
mov     [rbp+rdx+0A0h+S], al
mov     eax, [rbp+0A0h+j]
movsxd  rdx, eax
movzx   eax, [rbp+0A0h+temp_0]
mov     [rbp+rdx+0A0h+S], al
mov     eax, [rbp+0A0h+i]
cdq     [rbp+rax+0A0h+S]
movzx   edx, [rbp+0A0h+j]
mov     eax, [rbp+0A0h+j]
cdq     [rbp+rax+0A0h+S]
movzx   eax, [rbp+0A0h+j]
add     eax, edx
movzx   eax, al
mov     [rbp+0A0h+t], eax
mov     eax, [rbp+0A0h+k]
cdq     [rbp+0A0h+ciphertext]
add     rax, rdx
movzx   r8d, byte ptr [rax]
mov     eax, [rbp+0A0h+t]
cdq     [rbp+rax+0A0h+S]
movzx   ecx, [rbp+0A0h+k]
cdq     [rbp+0A0h+plaintext]
mov     rdx, [rbp+0A0h+plaintext]
add     rax, rdx
mov     edx, r8d
xor     edx, ecx
mov     [rax], dl
add     [rbp+0A0h+k], 1
```

**Perform XOR Operation with Ciphertext and Update Plaintext:**

```
mov     eax, [rbp+0A0h+k]        ; Load the value of k into EAX
cdq     [rbp+0A0h+k]            ; Sign-extend EAX to RAX
```

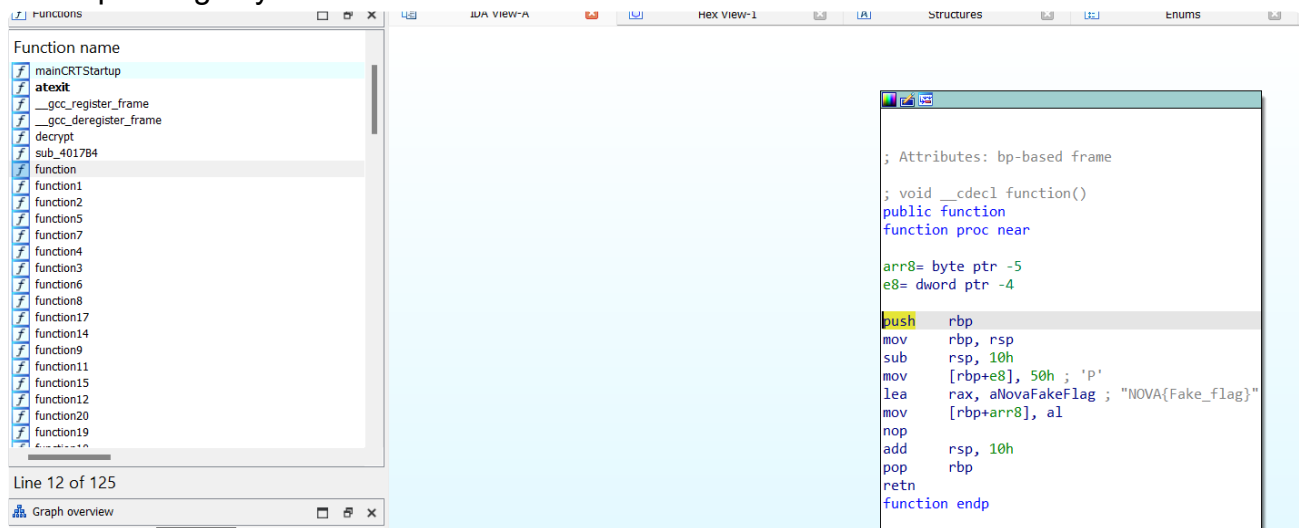


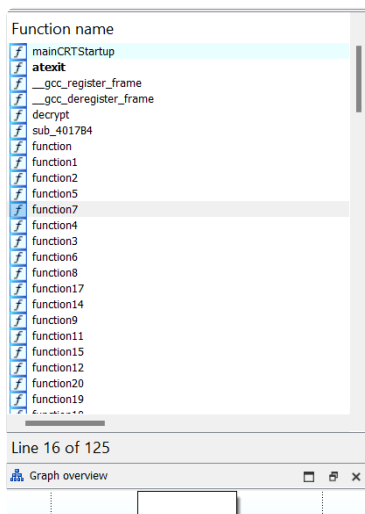
```

mov     rdx, [rbp+0A0h+ciphertext] ; Load the address of ciphertext into RDX
add     rax, rdx                    ; Add the offset to the base address of
ciphertext
movzx   r8d, byte ptr [rax]        ; Load the byte value from the ciphertext into
R8D
mov     eax, [rbp+0A0h+t]           ; Load the temporary value into EAX
cdqe                                         ; Sign-extend EAX to RAX
movzx   ecx, [rbp+rax+0A0h+S]       ; Load the value of S[t] into ECX
mov     eax, [rbp+0A0h+k]           ; Load the value of k into EAX
cdqe                                         ; Sign-extend EAX to RAX
mov     rdx, [rbp+0A0h+plaintext]   ; Load the address of plaintext into RDX
add     rax, rdx                    ; Add the offset to the base address of
plaintext
mov     edx, r8d                    ; Move the ciphertext byte value into EDX
xor     edx, ecx                    ; Perform the XOR

```

In the challenge, we encounter various functions, some of which contain fake flags. However, upon examining the code, we identify that the decryption algorithm being used is RC4. To progress and obtain the correct flag, we must locate the ciphertext and the corresponding key.





```

; Attributes: bp-based frame

; void __cdecl function7()
public function7
function7 proc near

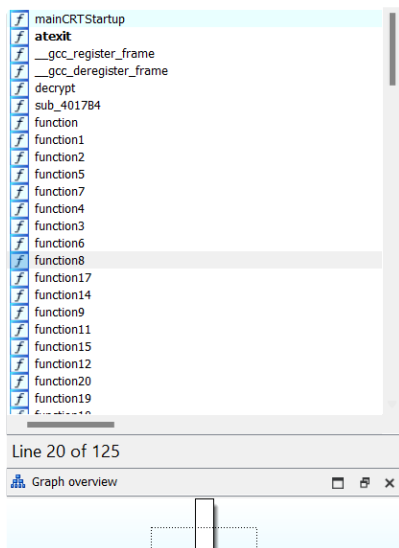
arr7= byte ptr -5
d7= dword ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+d7], 46h ; 'F'
lea     rax, aNovaSearchForD ; "NOVA{Search_for_decryption}"
mov     [rbp+arr7], al
nop
add     rsp, 10h
pop     rbp
retn
function7 endp

```

Also there was a hint "Search for decryption".

At function "8". You can see the "Secret" which is the key for decryption as "HAPPYHACKING"

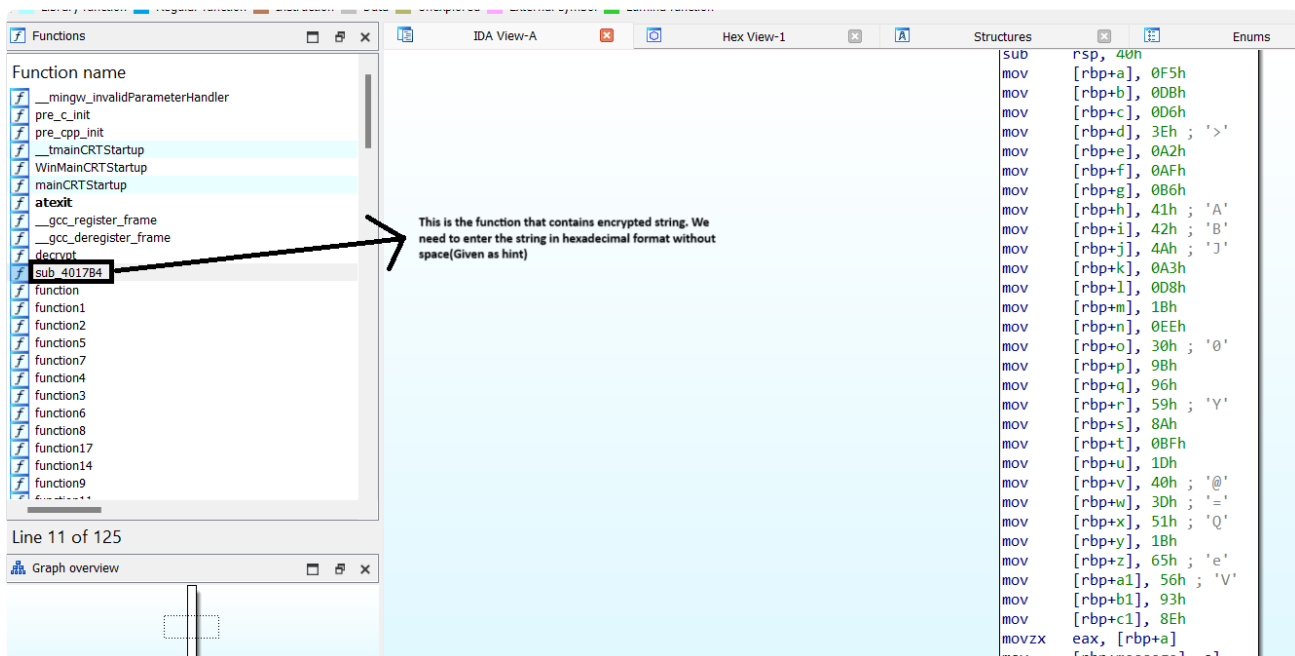


```

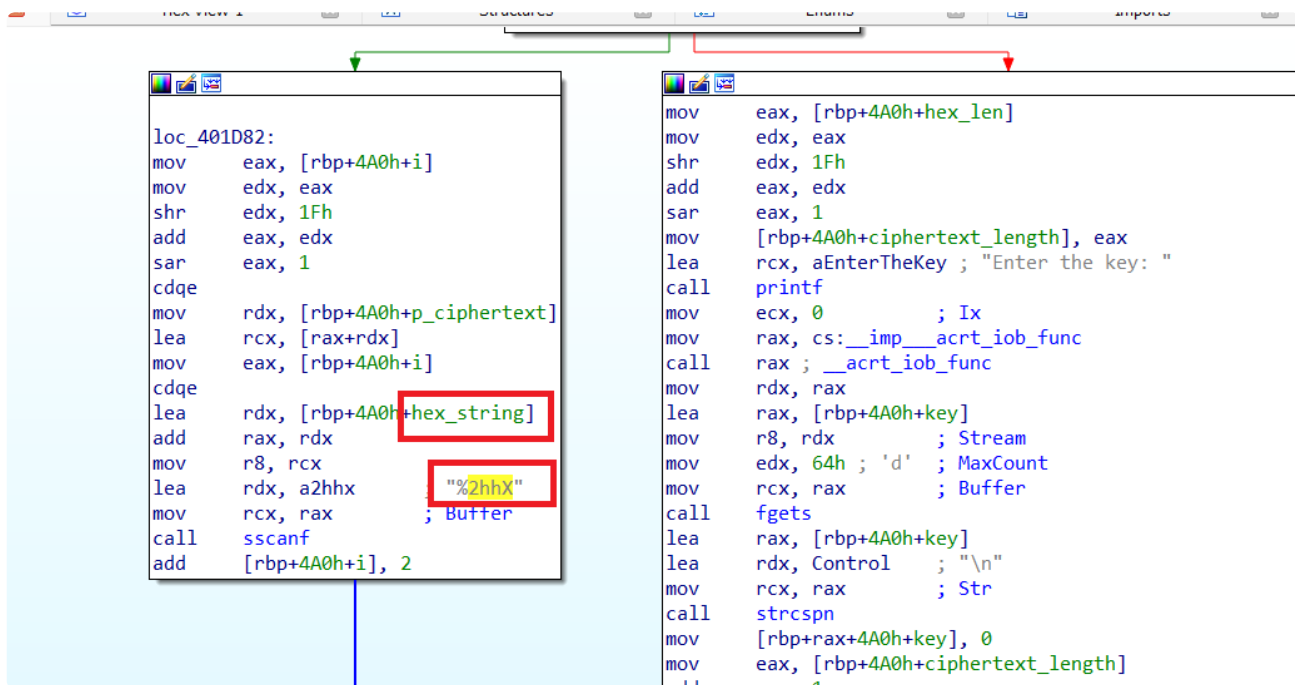
push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+A], 48h ; 'H'
mov     [rbp+B], 41h ; 'A'
mov     [rbp+C], 50h ; 'P'
mov     [rbp+D], 50h ; 'P'
mov     [rbp+E], 59h ; 'Y'
mov     [rbp+F], 48h ; 'H'
mov     [rbp+G], 41h ; 'A'
mov     [rbp+H], 43h ; 'C'
mov     [rbp+I], 48h ; 'K'
mov     [rbp+J], 49h ; 'I'
mov     [rbp+K], 4Eh ; 'N'
mov     [rbp+L], 47h ; 'G'
movzx   eax, [rbp+A]
mov     [rbp+secret], al
movzx   eax, [rbp+B]
mov     [rbp+secret+1], al
movzx   eax, [rbp+C]
mov     [rbp+secret+2], al
movzx   eax, [rbp+D]
mov     [rbp+secret+3], al
movzx   eax, [rbp+E]
mov     [rbp+secret+4], al
movzx   eax, [rbp+F]
mov     [rbp+secret+5], al

```

But still we need ciphertext to decrypt the plaintext and find the flag and lets look dig deeper in remaining functions.



By analysing the main function we can infer that it gets the encrypted string as hexadecimal value.



By Extracting all the hex values we can get the encrypted string:

F5 DB D6 3E A2 AF B6 41 42 4A A3 D8 1B EE 30 9B 96 59 8A BF 1D 40 3D 51 1B 65  
56 93 8E

It is mentioned as without space

```
F5DBD63EA2AFB641424AA3D81BEE309B96598ABF1D403D511B6556938E
```

We got the Encrypted string(hex) as

```
F5DBD63EA2AFB641424AA3D81BEE309B96598ABF1D403D511B6556938E
```

We already got key as

```
HAPPYHACKING
```

```
G:\>rev.exe
Welcome to NOVA-CTF2024
-----
Enter Encrypted string Without Space:F5DBD63EA2AFB641424AA3D81BEE309B96598ABF1D403D511B6556938E
Enter the key: HAPPYHACKING
Decrypted text: NOVA{Revers3_H4ck3r_Unl0cked}
```

Also we can decrypt online:

The screenshot shows the dcode.fr/rc4-cipher website. On the left, there is a search bar with the text "Search for a tool" and a search button. Below it, there is a section for "Results" showing the decrypted text: "NOVA{Revers3\_H4ck3r\_Unl0cked}". On the right, there is a section titled "RC4 CIPHER" with a subtitle "Cryptography · Modern Cryptography · RC4 Cipher". Below this, there is a section titled "RC4 DECODER" with a subtitle "★ TEXT/MESSAGE/CHARACTER STRING". The input field contains the hexadecimal string "F5 DB D6 3E A2 AF B6 41 42 4A A3 D8 1B EE 30 9B 96 59 8A BF 1D 40 3D 51 1B 65 56 93 8E". The output field shows the decrypted text "NOVA{Revers3\_H4ck3r\_Unl0cked}". Below the output field, there is a section titled "RC4 ENCRYPTION/DECRYPTION KEY" with a dropdown menu showing "HAPPYHACKING". Below this, there is a section titled "RESULTS FORMAT" with radio buttons for "ASCII (PRINTABLE) CHARACTERS", "HEXADECIMAL 00-7F-FF", "DECIMAL 0-127-255", "OCTAL 000-177-377", "BINARY 00000000-11111111", "INTEGER NUMBER", and "FILE TO DOWNLOAD". The "ASCII (PRINTABLE) CHARACTERS" option is selected. Below the format options, there is a button labeled "▶ DECRYPT/ENCRYPT".

If you need to know about RC4 encryption algorithm, please carefully read the instructions of this t

Input Content

F5DBD63EA2AFB641424AA3D81BEE309B96598ABF1D403D511B6556938E

Password HAPPYHACKING

Charset UTF-8 ▼

In-Format hex ▼

Out-Format string ▼

RC4 Encrypt

RC4 Decrypt

Copy

Clear

Output Result

NOVA{Revers3\_H4ck3r\_Unlocked}

Finally we found the flag as

NOVA{Revers3\_H4ck3r\_Unlocked}