

## 第一次作业

chatgpt简化后的比较key1的逻辑

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char key1[12];
    char key2[12];

    // 初始化v3
    char v3[11] = {0};
    char v7[12] = {0};

    // 对v3进行一系列操作
    for (int i = 0; i < 11; ++i) {
        v3[i] = v3[i] - 1;
    }

    // 对v7进行一系列操作
    int v8 = 0;
    int v9 = v3 - v7;
    do {
        v7[v8] = v8 * v8 + v7[v8 + v9] * v7[v8 + v9];
        ++v8;
    } while (v8 < 11);

    // 获取用户输入的key1
    printf("please input key1: ");
    scanf("%11s", key1);
}
```

```

// 对用户输入的key1进行一系列操作
for (int i = 0; i < 11; ++i) {
    key1[i] = i * i + key1[i] * key1[i];
}
key1[11] = 0;

// 比较v7和key1
if (strcmp(v7, key1) == 0) {
    // 进行相应操作
}

return 0;
}

```

观察逻辑可知v3和key1相等。

gpt的简化有一点小问题，ida中可以看到v3来自于0x40fa58 (xipbsfzpv@) 所有字符减1，也就是： **whoareyou?**

```

def decode(a1, a2):
    v6 = bytearray(11)

    if a2 == 1:
        v6 = bytes([byte ^ 0x77 for byte in a1])
    elif a2 == 2:
        v6 = bytes([(byte >> 4) + ((byte & 0xF) << 4) for byte in a1])
    else:
        v6 = bytes([byte - 1 for byte in a1])

    return v6

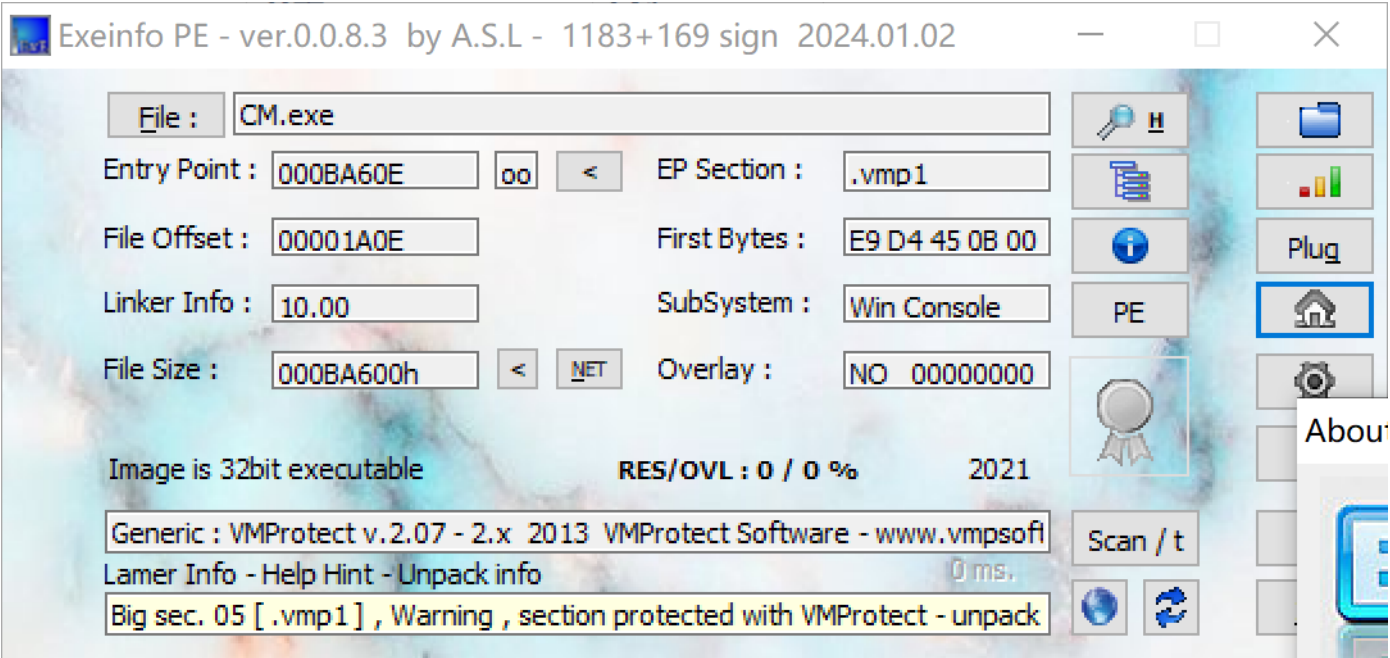
```

gpt给出的DLLU.dll中的decode算法

解出 `b'iamteacher'` 和 `b"don'tjoke!"`

第二次作业

带有vmp



vmp2.07，但其实关键逻辑没有v（异或和memcmp）。

调试检测

程序会检测自己是否在调试，这里直接过掉相关检测逻辑后attach上去。

大量花指令

日志	笔记	断点	内存布局	调用堆栈	SEH链	脚本	符号
•	00EE126E	33C0		xor eax, eax			
•	00EE1270	58		pop eax			
→	00EE1271	E8 CE060000		call cm.EE1944			
•	00EE1276	E8 EB10564D		call 4E442366			
•	00EE127B	50		push eax			
•	00EE127C	72 65		jb 551255			

花指令是没有意义的，这里没有恢复，所有的花直接单步过去。

CRC检测

程序应该存在使用CRC检测自己代码是否被修改，x32dbg中直接下软件断点不可用，必须使用硬件断点。

在olldb中通过FVMP插件找到OEP，此时程序的各个段是解密的并且程序刚开始，我们将每个段dump出来。

00EE0000	00001000	CM		PE 文件头	Image	R	RWE
00EE1000	0000A000	CM	.text	代码	Image	R	RWE
00EEB000	00003000	CM	.rdata	数据	Image	R	RWE
00EEE000	00004000	CM	.data		Image	R	RWE
00EF2000	000A7000	CM	.vmp0		Image	R	RWE
00F99000	000BA000	CM	.vmp1	SFX, 输入表	Image	R	RWE
01053000	00001000	CM	.vmp2		Image	R	RWE

把所有的dump文件合并起来，放到ida中观察字符串。

[s]	.vmp1:00...	0000000A	C	EaSyStEp1
[s]	.vmp1:00...	0000001B	C	Success Step 1!\n Next Key:
[s]	.vmp1:00...	00000009	C	failed!\n
[s]	.vmp1:00...	0000000A	C	EaSyStEp2
[s]	.vmp1:00...	0000001B	C	Success Step 2!\n Next Key:
[s]	.vmp1:00...	0000000A	C	EaSyStEp3
[s]	.vmp1:00...	00000015	C	Success Step 3!\n OK!

输入 EaSyStEp1 可以过第一步，但是EaSyStEp2过不了第二步。

[s]	.vmp1:00...	00000041	C	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
-----	-------------	----------	---	---

看到了base64表，将EaSyStEp2进行base64编码， RWFTeVN0RXAy 可以过第二步。

[s]	.vmp1:00...	0000000A	C	EaSyStEp1
[s]	.vmp1:00...	0000001B	C	Success Step 1!\n Next Key:
[s]	.vmp1:00...	00000009	C	failed!\n
[s]	.vmp1:00...	0000000A	C	EaSyStEp2
[s]	.vmp1:00...	0000001B	C	Success Step 2!\n Next Key:
[s]	.vmp1:00...	0000000A	C	EaSyStEp3
[s]	.vmp1:00...	00000015	C	Success Step 3!\n OK!
[s]	.vmp1:00...	0000001B	C	abcdefghijklmnopqrstuvwxyz
[s]	.vmp1:00...	0000001B	C	ABCDEFGHIJKLMNOPQRSTUVWXYZ
[s]	.vmp1:00...	0000001B	C	abcdefghijklmnopqrstuvwxyz
[s]	.vmp1:00...	0000001B	C	ABCDEFGHIJKLMNOPQRSTUVWXYZ
[s]	.vmp1:00...	0000014D	C	2eiT5t129FmB6cv///+jY4PDArhhAAAAD7cTagIDHCRcD7c5MddmiTmDwQJIhcAPheP///815ckNSw1LDSuEzrUO0YWBtQrnBmxUeN8K9wVAT2Z+m...
[s]	.vmp1:00...	00000009	C	fseewefa
[s]	.vmp1:00...	00000011	C	16AEAEFstIKLLl11
[s]	.vmp1:00...	0000000E	C	DecodePointer

在 Success Step 3!\n 附近有一些关键字符串，其中 fseewefa 与 EaSyStEp3 的长度接近。

怀疑是循环异或，但是异或的输入有不可见字符，只能调试去看具体的逻辑了。

输入：

[s]	.vmp1:00...	00000000	C	
-----	-------------	----------	---	--

1234567890

eax是key3的长度。

Address	Disassembly	Comment
00EE1CC1	837E 00 00	cmp dword ptr ds:[esi+8],0
00EE1CC5	75 07	jne cm.EE1CCE
00EE1CC7	56	push esi
00EE1CC8	E8 E5260000	call cm.EE43B2
00EE1CCD	59	pop ecx
00EE1CCE	8806	mov eax,dword ptr ds:[esi]
00EE1CD0	3B46 08	cmp eax,dword ptr ds:[esi+8]
00EE1CD3	75 09	jne cm.EE1CDE
00EE1CD5	837E 04 00	cmp dword ptr ds:[esi+4],0
00EE1CD9	75 C9	jne cm.EE1CA4
00EE1CDB	40	inc eax
00EE1CDE	8906	mov dword ptr ds:[esi],eax
00EE1CE0	FF0E	dec dword ptr ds:[esi]
00EE1CE4	F646 0C 40	test byte ptr ds:[esi+C],40
00EE1CE6	8806	mov eax,dword ptr ds:[esi]
00EE1CE8	74 09	je cm.EE1CF1
00EE1CEA	3818	cmp byte ptr ds:[eax],bl
00EE1CEC	74 07	je cm.EE1CF3
00EE1CED	40	inc eax
00EE1CEF	8906	mov dword ptr ds:[esi],eax
00EE1CF1	EB B3	jmp cm.EE1CA4
00EE1CF3	8818	mov byte ptr ds:[eax],bl
00EE1CF5	8846 0C	mov eax,dword ptr ds:[esi+C]
00EE1CF7	F446 04	inc dword ptr ds:[esi+4]
00EE1CF9	83E0 EF	and eax,FFFFFFF
00EE1CFC	83C8 01	or eax,1
00EE1CFF	8946 0C	mov dword ptr ds:[esi+C],eax
00EE1D02	8BC3	mov eax,ebx
00EE1D04	25 FF000000	and eax,FF
00EE1D09	98 98	jmp cm.EE1CA6
00EE1D0B	8BF8	mov edi,edi
00EE1D0D	55	push ebp
00EE1D0E	8BEC	mov ebp,esp
00EE1D10	53	push ebx
00EE1D11	56	push esi
00EE1D12	8B75 08	mov esi,dword ptr ss:[ebp+8]
00EE1D15	8846 0C	mov eax,dword ptr ds:[esi+C]
00EE1D18	8BC8	mov ecx,eax
00EE1D1A	80E1 03	and cl,3

esi+8是输入的 1234567890 这里是判断key3字符串的长度。

Debugger screenshot showing assembly code and registers. The assembly code includes instructions like `push ebp`, `mov ebp, esp`, `push ebx`, `push esi`, `mov eax, cm.EEF810`, `push edi`, `lea edx, dword ptr ds:[eax+1]`, `mov edi, edi`, `mov cl, byte ptr ds:[eax]`, `inc eax`, `test cl, cl`, `jnz cm.EE10F0`, `sub eax, edx`, `xor esi, esi`, `mov ebx, eax`, `cmp dword ptr ss:[ebp+10], esi`, `jle cm.EE1167`, `mov edi, dword ptr ss:[ebp+8]`, `sub edi, dword ptr ss:[ebp+8]`, `push eax`, `mov eax, ebx`, `xor eax, eax`, `inc cm.EE1108`, `call B954616E`, `jmp cm.EE1110`, `inc cm.EE1116`, `call 91F57979`, `call 908F938`, `add byte ptr ds:[eax], al`, `call ECAB777F`, `add eax, FA74C033`, `call B46D016C`, `inc ebp`, `or al, 80`, `or al, 0`, `mov eax, esi`, `cq`.

Registers: EAX: 0000000A, EBX: 00000000, ECX: 00AFFB0C, EDX: 00AFFB0C, ESP: 00AFFAD4, ESI: 00EE134D, EDI: 00000000, EIP: 00EE10E5, EFLAGS: 00000216, ZF: 0, PF: 1, AF: 1, OF: 0, SF: 0, DF: 0, CF: 0, TF: 0, IF: 1.

Stack (stdcall): 1: [esp+4] 00000000 00000000, 2: [esp+8] 00AFFE28 00AFFE28, 3: [esp+C] 00EE13CD cm.00EE13CD, 4: [esp+10] 00AFFD0C 00AFFD0C, 5: [esp+14] 00AFFD0C 00AFFD0C.

Debugger screenshot showing assembly code and registers. The assembly code includes instructions like `push ebp`, `mov ebp, esp`, `push ebx`, `push esi`, `mov eax, cm.EEF810`, `push edi`, `lea edx, dword ptr ds:[eax+1]`, `mov edi, edi`, `mov cl, byte ptr ds:[eax]`, `inc eax`, `test cl, cl`, `jnz cm.EE10F0`, `sub eax, edx`, `xor esi, esi`, `mov ebx, eax`, `cmp dword ptr ss:[ebp+10], esi`, `jle cm.EE1167`, `mov edi, dword ptr ss:[ebp+8]`, `sub edi, dword ptr ss:[ebp+8]`, `push eax`, `mov eax, ebx`, `xor eax, eax`, `inc cm.EE1108`, `call B954616E`, `jmp cm.EE1110`, `inc cm.EE1116`, `call 91F57979`, `call 908F938`, `add byte ptr ds:[eax], al`, `call ECAB777F`, `add eax, FA74C033`, `call B46D016C`, `inc ebp`, `or al, 80`, `or al, 0`, `mov eax, esi`, `cq`.

Registers: EAX: 0000000A, EBX: 00000000, ECX: 00AFFB0C, EDX: 00AFFB0C, ESP: 00AFFAD4, ESI: 00EE134D, EDI: 00000000, EIP: 00EE10F9, EFLAGS: 00000202, ZF: 0, PF: 0, AF: 0, OF: 0, SF: 0, DF: 0, CF: 0, TF: 0, IF: 1.

Stack (stdcall): 1: [esp+4] 00EE134D cm.00EE134D, 2: [esp+8] 00000000 00000000, 3: [esp+C] 00AFFE28 00AFFE28, 4: [esp+10] 00EE13CD cm.00EE13CD, 5: [esp+14] 00AFFD0C 00AFFD0C.

到这里用上了fseewefa。

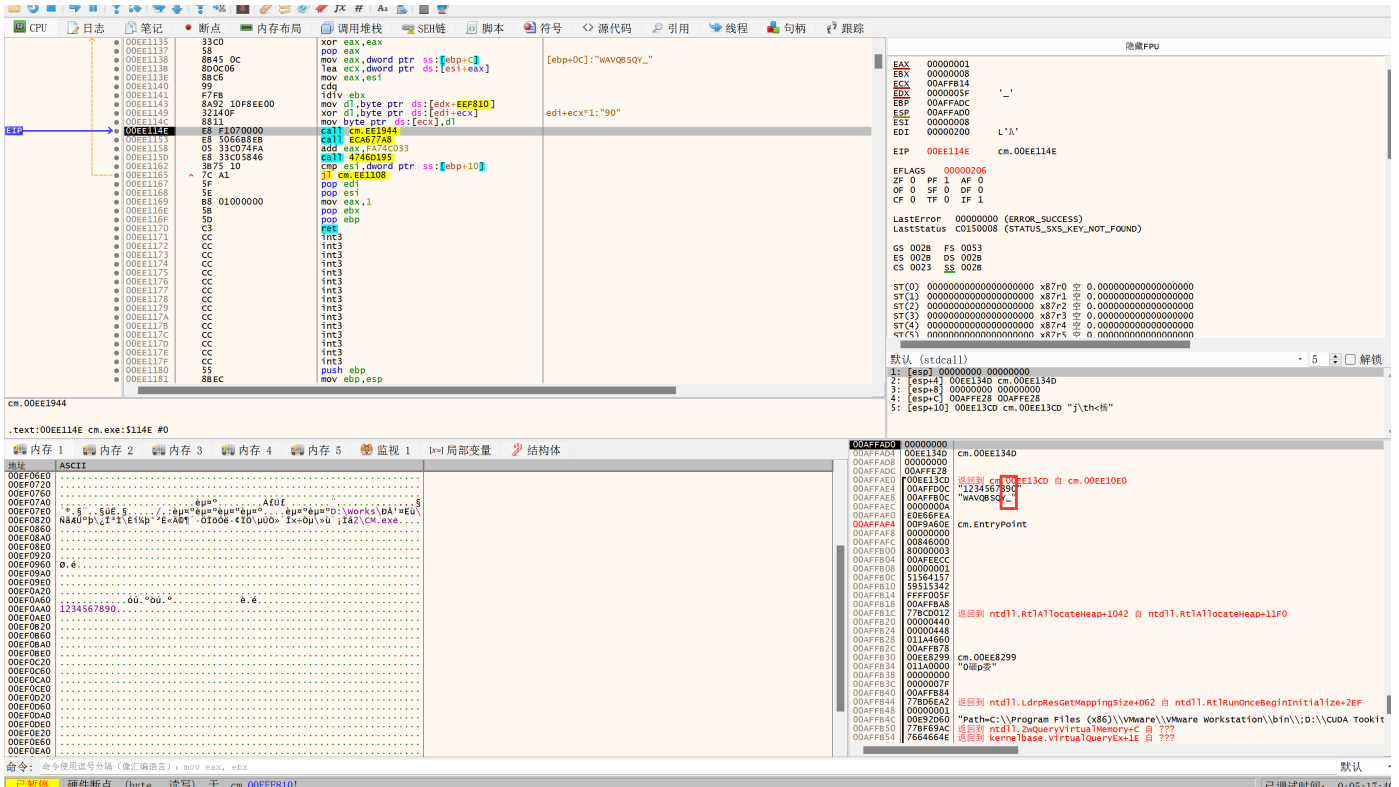
Debugger screenshot showing assembly code and registers. The assembly code includes instructions like `push ebp`, `mov ebp, esp`, `push ebx`, `push esi`, `mov eax, cm.EEF810`, `push edi`, `lea edx, dword ptr ds:[eax+1]`, `mov edi, edi`, `mov cl, byte ptr ds:[eax]`, `inc eax`, `test cl, cl`, `jnz cm.EE10F0`, `sub eax, edx`, `xor esi, esi`, `mov ebx, eax`, `cmp dword ptr ss:[ebp+10], esi`, `jle cm.EE1167`, `mov edi, dword ptr ss:[ebp+8]`, `sub edi, dword ptr ss:[ebp+8]`, `push eax`, `mov eax, ebx`, `xor eax, eax`, `inc cm.EE1108`, `call B954616E`, `jmp cm.EE1110`, `inc cm.EE1116`, `call 91F57979`, `call 908F938`, `add byte ptr ds:[eax], al`, `call ECAB777F`, `add eax, FA74C033`, `call B46D016C`, `inc ebp`, `or al, 80`, `or al, 0`, `mov eax, esi`, `cq`.

Registers: EAX: 00000008, EBX: 00000000, ECX: 00AFFB0C, EDX: 00EEF811, ESP: 00AFFAD4, ESI: 00EE134D, EDI: 00000000, EIP: 00EE10F9, EFLAGS: 00000202, ZF: 0, PF: 0, AF: 0, OF: 0, SF: 0, DF: 0, CF: 0, TF: 0, IF: 1.

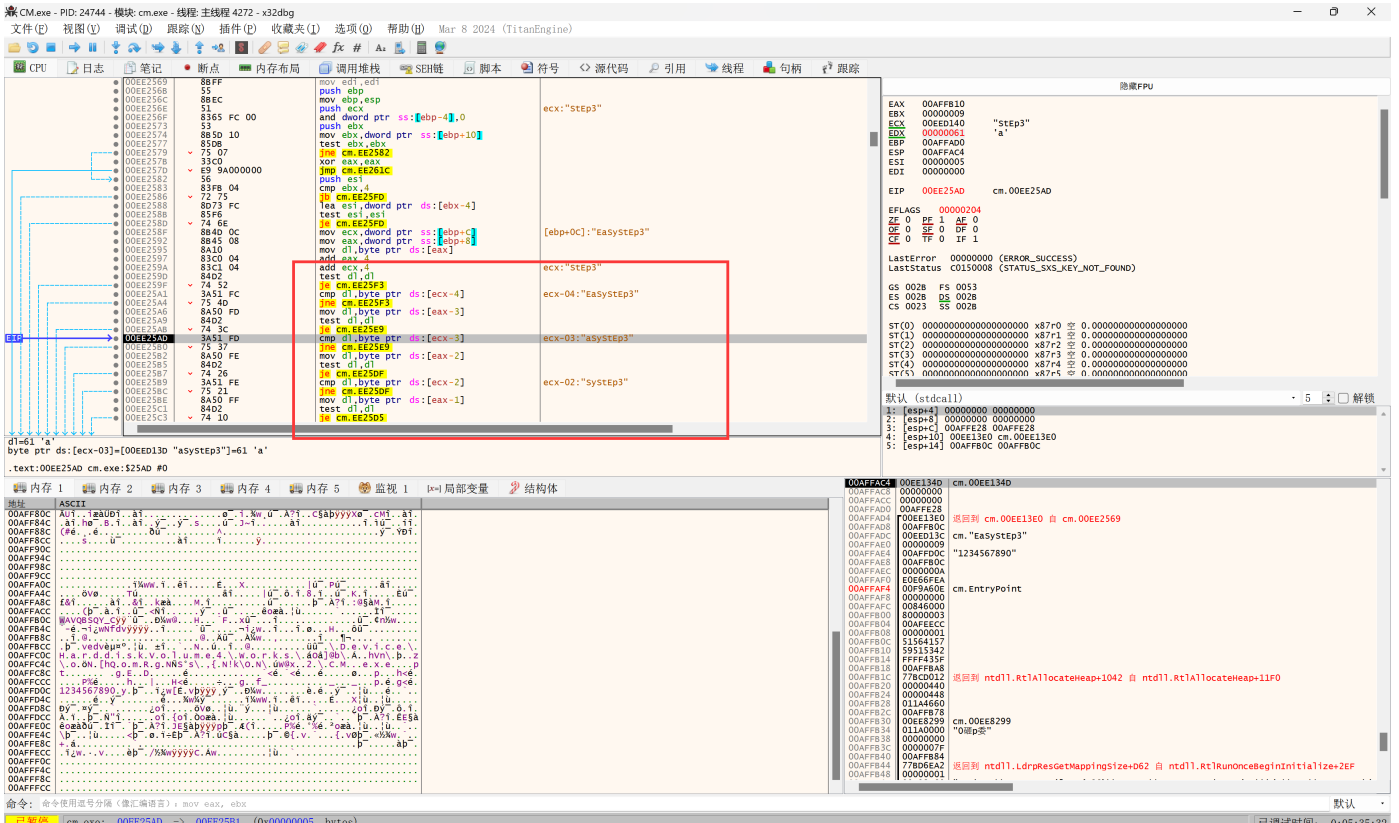
Stack (stdcall): 1: [esp+4] 00EE134D cm.00EE134D, 2: [esp+8] 00000000 00000000, 3: [esp+C] 00AFFE28 00AFFE28, 4: [esp+10] 00EE13CD cm.00EE13CD, 5: [esp+14] 00AFFD0C 00AFFD0C.

用 fseewefa 与 1234567890 进行循环异或。





将key3和异或结果以及key3的长度压入栈中，并调用一个关键函数（猜测为memcmp）。



这里将异或结果与 **EaSyStEp3** 进行比较，我们手动修改异或结果使得memcmp返回0，最后 step3 success。

作业中给出的提交方式是在cmd中直接输入字符，但其实字符在包括不可见字符，在terminal中无法输入不可见字符，可以先将key写入到文本中，在记事本用UTF-8打开，复制到Windows的原始cmd中，即可实现作业中展示的输入方式。

```
strings = ["EaSyStEp1", "RWFTeVN0RXAy", "#\x126\x1c$\x11#\x11U"]
```

```
# 将字符串写入文件
```

```
with open("exp.txt", "wb") as file:
```

```
    for string in strings:
```

```
        file.write(string + "\n")
```

```
C:\Users\eqqre>D.
```

```
D. >CM.exe
```

```
Input Key1: EaSyStEp1
```

```
Success Step 1!
```

```
Next Key:RWFTeVN0RXAy
```

```
Success Step 2!
```

```
Next Key:#^R6^\$^Q#^QU
```

```
Success Step 3!
```

```
OK!
```