Cipher 2

- Crib: The first Character of the plaintext is R
- Based on this we can assume that the plaintext is constructed from the Latin Alphabet
- Based on the context of the course we can assume that the message is English

In [1]:
```python
## import message
def format_hex_display(data, bytes_per_row=16):
    formatted_hex = ""
    for i in range(0, len(data), bytes_per_row * 2):
        # Grab a slice of the hex string corresponding to the number of bytes per row
        row = data[i:i + bytes_per_row * 2]
        # Break the row into byte-pairs (2 characters each)
        byte_pairs = ' '.join(row[j:j+2] for j in range(0, len(row), 2))
        formatted_hex += byte_pairs + '\n'
    return formatted_hex

with open('21.hex', 'rb') as file:
    ciphertext = file.read()

    # convert to list of bytes for easy processing
    c_bytes = list(ciphertext)


# Print formatted hex values
print(format_hex_display(ciphertext.hex()))
```

```
3c 91 1d 91 0f 86 0d 9c 0b 86 1d d4 07 9a 4e c5
57 c6 57 d4 0f 80 4e a4 1c 9d 00 97 0b 80 01 9a
4e a1 00 9d 18 91 1c 87 07 80 17 d4 1a 81 1c 9a
0b 90 4e 95 4e 98 07 82 07 9a 09 d4 0d 95 1a d4
07 9a 1a 9b 4e 95 4e 80 0b 98 0b 84 06 9b 00 91
63 fe
```

- We can make an initial assumption that 3C equates to R

In [2]:
```python
## range analysis

# highest hex number
max_hex = max(c_bytes)
print("Max hex value: ", hex(max_hex))

# lowest hex number
min_hex = min(c_bytes)
print("Min hex value: ", hex(min_hex))

# decode hex file to string with different character sets
charsets = ['utf-8', 'latin1', 'ascii', 'cp1252', 'utf-16', 'utf-32']
for charset in charsets:
    try:
        ciphertext_str = ciphertext.decode(charset)
        print(f"Decoded with {charset}: {ciphertext_str}")
    except Exception as e:
        print(f"Failed to decode with {charset}: {e}")
```

```
Max hex value:  0xfe
Min hex value:  0x0
Failed to decode with utf-8: 'utf-8' codec can't decode byte 0x91 in position 1: invalid start byte
⍰⍰Ô⍰⍰⍰⍰N⍰N⍰⍰
⍰⍰
⍰⍰⍰⍰cþ⍰
⍰⍰⍰N¡⍰⍰⍰⍰
⍰⍰⍰⍰Ô⍰⍰⍰
⍰⍰
⍰N⍰N⍰⍰⍰⍰⍰        Ô
Failed to decode with ascii: 'ascii' codec can't decode byte 0x91 in position 1: ordinal not in range(128)
Failed to decode with cp1252: 'charmap' codec can't decode byte 0x9d in position 25: character maps to <undefined>
Decoded with utf-16: 鄼鄝蘏鰍蘋퐝騇앎왗푗耏ꑎ鴜需耋騁ꅎ鴀鄘蜜耇퐗脚騜逋镎顎與騇퐉锍퐚騇鬚镎聎頋萋鬆鄀﹣
Failed to decode with utf-32: 'utf-32-le' codec can't decode bytes in position 0-3: code point not in range(0x110000)
```

- Cannot decode message to known character set, due to range problems and bytes equating to non printable characters
- With no knowledge of the structure or character mappings, pursuring freequency analysis methods is not plausisble
- Assuming that 3C = R, but the hex for R is 0x54. Need operation to convert them which can be mapped to all bytes
- The lecture slides discuss the XOR Cipher and OTP which both use the XOR Operation.
- If XOR has been used, we can XOR 3C and R to obtain a key

In [3]:
```python
# XOR the first byte with 'R' to get the key
key = c_bytes[0] ^ ord('R')
print("Key (hex):", hex(key))
```

```
# XOR every byte with the key
decrypted_bytes = [byte ^ key for byte in c_bytes]

# Print raw Ascii for decrypted bytes
print("Decrypted Message:", ''.join(chr(byte) for byte in decrypted_bytes))
```

```
Key (hex): 0x6e
Decrypted Message: Rÿsÿaècòeèsºiô «9¨9ºaî Êrónùeîoô Ïnóvÿréiîyºtïrôeþ û öiìiôgºcûtºiôtõ û îeöeêhõnÿ
```

- By applying the key to the ciphertext, we do not get an English message and there are many non-latin/english alphabet characters
- However there are Latin/English characters present, specifically every other character.

In [4]:
```
# strip non-printable characters

# ascii values for printable characters are 32-126
# print ascii for each byte in decrypted - if no char then print '-'
decrypted_string = ''.join(chr(byte) if 32 <= byte <= 126 else '-' for byte in decrypted_bytes)
print("Decrypted Message (stripped):", decrypted_string)
```

```
Decrypted Message (stripped): R-s-a-c-e-s-i- -9-9-a- -r-n-e-o- -n-v-r-i-y-t-r-e- - -i-i-g-c-t-i-t- - -e-e-h-n---
```

- By filtering out non-latin characters, the text looks more like English, with apropriate words lengths and characters.
- This suggests that the initial key is used to encrypt every other character and that another key is used for the remaining
- There are single character words present. In english, these can only be i or a

In [5]:
```
# find index of single character words
words = decrypted_string.split()
single_char_words = {}
i = 0
for word in words:
    if len(word) == 1:
        single_char_words[i] = word
    i += (len(word) + 1)
print("Single Character Words:", single_char_words)

# find potential second key
potential_keys = []
for i in single_char_words.keys():
    potential_keys.append(c_bytes[i] ^ ord('i'))
    potential_keys.append(c_bytes[i] ^ ord('a'))

# remove duplicates
potential_keys = list(set(potential_keys))
print("Potential Keys (hex):", [hex(key) for key in potential_keys])
```

```
Single Character Words: {51: '-', 69: '-'}
Potential Keys (hex): ['0xfc', '0xf4']
```

- The potential second keys are obtained by XORing the hex values of the single character words with i and a
- Giving us two potential keys

In [6]:
```
# print potential decryptions with second key - only printable ascii characters
print("Potential Decryptions:")
for i, k in enumerate(potential_keys):
    decrypted_key2 = [byte ^ k for byte in c_bytes]
    decrypted_string2 = ''.join(chr(byte) if 32 <= byte <= 126 else '-' for byte in decrypted_key2)
    print(" ", hex(k) , ":" ,decrypted_string2)

# choose second key
key_2 = potential_keys[1]
```

```
Potential Decryptions:
  0xfc : -m-m-z-`-z-(-f-9-:-(-|-X-a-k-|-f-]-a-m-{-|-(-}-f-l-i-d-~-f-(-i-(-f-g-i-|-d-x-g-m--
  0xf4 : -e-e-r-h-r- -n-1-2- -t-P-i-c-t-n-U-i-e-s-t- -u-n-d-a-l-v-n- -a- -n-o-a-t-l-p-o-e--
```

- The second decryption looks plausible. It is entirely made up of English letters

In [7]:
```
# decrypt even indexes with key 1 and odd indexes with key 2
decrypted = []
for i, byte in enumerate(c_bytes):
    if i % 2 == 0:
        decrypted.append(byte ^ key)
    else:
        decrypted.append(byte ^ key_2)

# print decrypted message
decrypted_string = ''.join(chr(byte) if 32 <= byte <= 126 else '-' for byte in decrypted)
print("Decrypted Message:", decrypted_string)
```

```
Decrypted Message: Researchers in 1929 at Princeton University turned a living cat into a telephone--
```

- Message is succesfully decrypted... almost

- Trailing non-printable characters

In [8]:
```
## sanity check
def fmt(byte):
    if byte < 10:
        return "  " + str(byte)
    elif byte < 100:
        return " " + str(byte)
    else:
        return str(byte)

print("Hx",":"," C ",":"," P ",":","Ascii")
for i in range(0, len(c_bytes)):
    hex_str = hex(c_bytes[i])[2:]
    if len(hex_str) == 1:
        hex_str = "0" + hex_str
    ds = decrypted_string[i]
    if ds == " ":
        ds = "space"
    print(hex_str,":",fmt(c_bytes[i]),":",fmt(decrypted[i]),":", ds)

## Why are there 2 non decrypted characters at the end?
```

```
Hx :  C  :  P  : Ascii
3c :  60 :  82 : R
91 : 145 : 101 : e
1d :  29 : 115 : s
91 : 145 : 101 : e
0f :  15 :  97 : a
86 : 134 : 114 : r
0d :  13 :  99 : c
9c : 156 : 104 : h
0b :  11 : 101 : e
86 : 134 : 114 : r
1d :  29 : 115 : s
d4 : 212 :  32 : space
07 :   7 : 105 : i
9a : 154 : 110 : n
4e :  78 :  32 : space
c5 : 197 :  49 : 1
57 :  87 :  57 : 9
c6 : 198 :  50 : 2
57 :  87 :  57 : 9
d4 : 212 :  32 : space
0f :  15 :  97 : a
80 : 128 : 116 : t
4e :  78 :  32 : space
a4 : 164 :  80 : P
1c :  28 : 114 : r
9d : 157 : 105 : i
00 :   0 : 110 : n
97 : 151 :  99 : c
0b :  11 : 101 : e
80 : 128 : 116 : t
01 :   1 : 111 : o
9a : 154 : 110 : n
4e :  78 :  32 : space
a1 : 161 :  85 : U
00 :   0 : 110 : n
9d : 157 : 105 : i
18 :  24 : 118 : v
91 : 145 : 101 : e
1c :  28 : 114 : r
87 : 135 : 115 : s
07 :   7 : 105 : i
80 : 128 : 116 : t
17 :  23 : 121 : y
d4 : 212 :  32 : space
1a :  26 : 116 : t
81 : 129 : 117 : u
1c :  28 : 114 : r
9a : 154 : 110 : n
0b :  11 : 101 : e
90 : 144 : 100 : d
4e :  78 :  32 : space
95 : 149 :  97 : a
4e :  78 :  32 : space
98 : 152 : 108 : l
07 :   7 : 105 : i
82 : 130 : 118 : v
07 :   7 : 105 : i
9a : 154 : 110 : n
09 :   9 : 103 : g
d4 : 212 :  32 : space
0d :  13 :  99 : c
95 : 149 :  97 : a
1a :  26 : 116 : t
d4 : 212 :  32 : space
07 :   7 : 105 : i
9a : 154 : 110 : n
1a :  26 : 116 : t
9b : 155 : 111 : o
4e :  78 :  32 : space
95 : 149 :  97 : a
4e :  78 :  32 : space
80 : 128 : 116 : t
0b :  11 : 101 : e
98 : 152 : 108 : l
0b :  11 : 101 : e
84 : 132 : 112 : p
06 :   6 : 104 : h
9b : 155 : 111 : o
00 :   0 : 110 : n
91 : 145 : 101 : e
63 :  99 :  13 : -
fe : 254 :  10 : -
```

In [ ]:
```python
# final two decrypted characters in decimal: 10 13
# 10: newline
print("\\n == " + str(ord("\n")))
```

```
# 13: carriage return
print("\\r == " + str(ord("\r")))

# Therefore final message is
decrypted_final = decrypted_string[:-2] + "\r\n"
print("Decrypted Message (final):", decrypted_final + "this should continue onto the next line")

print("Final Key:", hex(key)[2:] + hex(key_2)[2:])
# le boom
```

```
\n == 10
\r == 13
Decrypted Message (final): Researchers in 1929 at Princeton University turned a living cat into a telephone
this should continue onto the next line
Final Key:  6ef4
```

- By examining the ascii values of the final two non-printable characters we can see that they encode a new line.
- Official CR LF which is a common way to denote a new line in text

Success!