推测明文的过程：

```
# syml @ SYMLArch in ~/Temp/Crypto/home1 [11:57:25] C:130
$ python solve1-Interactive.py
##### Interactive #####
##### pos 0 #####
0 1 2 3 4 5 6 7 8 9 10
* * * * * * * * * w *
* * * * * * * * * e *
* * * * * * * * * t *
* * * * * * * * * t *
* * * * * * * * * y *
* * * * * * * * * t *
* * * * * * * * * t *
* * * * * * * * * w *
* * * * * * * * * a *
w e t t y t t w a * t
* * * * * * * * * t *
find any byte is space? y/Ny
which cipher text?[num]9
current:
 ['W', 'E', 'T', 'T', 'Y', 'T', 'T', 'W', 'A', ' ', 'T']
##### pos 1 #####
0 1 2 3 4 5 6 7 8 9 10
* * * * * * * * E * *
* * * * * * * * U * *
* * * * * * * * H * *
* * * * * * * * H * *
* * * * * * * * O * *
* * * * * * * * H * *
* * * * * * * * H * *
* * * * * * * * E * *
E U H H O H H E * t H
* * * * * * * * t * *
* * * * * * * * H * *
find any byte is space? y/Ny
which cipher text?[num]8
current:
 ['We', 'Eu', 'Th', 'Th', 'Yo', 'Th', 'Th', 'We', 'A ', ' T', 'Th']
##### pos 2 #####
0 1 2 3 4 5 6 7 8 9 10
* L E E U E E * * H E
L * * * * * * L D * *
```

根据推断出的明文片段推测、搜索到对应的原文，解密：

```
# syml @ SYMLArch in ~/Temp/Crypto/home1 [12:21:42] C:1
$ python solve1-finally.py
key is:
66396ec289c389c39bc398c38cc29874352ac38d63c295102ec2afc38e78c2aa7fc3ad28c2a07f6bc389c28d29c3850b69c2b033c29a19c3b8c2aa
401ac29c6d70c28fc280c38066c38763c3bec3b0123148c38dc398c3a802c3905bc2a9c28777335dc2aec3bcc3acc395c29c433a6b26c28b60c2bf
4ec3b03cc29a6110c298c2bb3ec29a3161c3adc387c2b804c2a33522c38fc39202c392c386c28c57376ec39bc2a8c382c38a50027c61246cc3a2c2
a12b0c4502175010c380c2a1c2ba4625786dc2911100797dc28a47c3a9c28b0204c384c3af06c38867c2a950c3b11ac389c289c39ec2a8c28fc391
c39bc3b1674874c29ec394c386c3b45b384cc29dc296c38411

cipher 8 is:
We can see the point where the chip is unhappy if a wrong bit is sent and consumes more power from the environment - A
di Shamir

# syml @ SYMLArch in ~/Temp/Crypto/home1 [12:21:53]
```

```
key:
66396ec289c389c39bc398c38cc29874352ac38d63c295102ec2afc38e78c2aa7fc3ad28c2a07f6bc389c28d29c3850b69c2b033c29a19c3b8c2aa401ac29c6d70c28fc280c38066c38763c3bec3b0123148c38dc398c3a802c3905bc2a9c28777335dc2aec3bcc3acc395c29c433a6b26c28b60c2bf4ec3b03cc29a6110c298c2bb3ec29a3161c3adc387c2b804c2a33522c38fc39202c392c386c28c57376ec39bc2a8c382c38a50027c61246cc3a2c2a12b0c4502175010c380c2a1c2ba4625786dc2911100797dc28a47c3a9c28b0204c384c3af06c38867c2a950c3b11ac389c289c39ec2a8c28fc391c39bc3b1674874c29ec394c386c3b45b384cc29dc296c38411
plaintext:
We can see the point where the chip is unhappy if a wrong bit is sent and consumes more power from the environment - Adi Shamir
```

```python
# 推断 solve1-Interactive.py
from binascii import a2b_hex

hex_c = [

 b"315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e",

 b"234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f",

 b"32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb",

 b"32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa",

 b"3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070",

 b"32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4",

 b"32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d96815969347 77e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce",

 b"315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3",

 b"271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027",

 b"466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83",
```

```
 b"32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904"
]

c = []
for i in hex_c:
    c.append(a2b_hex(i))

m = ["" for i in range(len(c))]

avavil_char = "ABCDEFGHIJKLMNOPQRSTUVXYZabcdefghijklmnopqrstuvwxyz ,.'"

print("##### Interactive #####")
for i in range(30):
    print(f"##### pos {i} #####")
    for j in range(len(c)):
            print(f"{j} ", end="")
    print()
    for j in range(len(c)):
        for k in range(len(c)):
            if chr(c[k][i] ^ c[j][i]) in avavil_char:
                print(f"{chr(c[k][i] ^ c[j][i])} ", end="")
            else:
                print("* ", end="")
        print()
    ch = input("find any byte is space? y/N")
    if ch == "y":
        ind = int(input("which cipher text?[num]"))
        m[ind] += " "
        for j in range(len(c)):
            if j == ind:
                continue
            m[j] += chr(c[j][i] ^ c[ind][i] ^ ord(" "))
    else:
        for j in range(len(c)):
            m[j] += "*"
    print("current:\n",m)


'''select like this
##### pos 16 #####
0 1 2 3 4 5 6 7 8 9 10
* * * * * * * * E * *
* * * * * * * * A * *
* * * * * * * * B * *
* * * * * * * * R * *
* * * * * * * * O * *
* * * * * * * * P * *
* * * * * * * * P * *
* * * * * * * * O * *
E A B R O P P O * O G
* * * * * * * * O * *
* * * * * * * * G * *
find any byte is space? y/Ny <-- user input
which cipher text?[num]8 <-- user input

```

```
'''
```

```
# decrypt
# search on google
# text 6 is: There are two types of cyptography: one that allows the Government to use brute force to break the code, and one that requires the Government to use brute force to break you.
from binascii import a2b_hex, b2a_hex

hex_c = [

 b"315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e",

 b"234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f",

 b"32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb",

 b"32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa",

 b"3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070",

 b"32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4",

 b"32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce",

 b"315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3",

 b"271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027",
```

```
 b"466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83",
 b"32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904"
]

c = []
for i in hex_c:
    c.append(a2b_hex(i))

key = []
plain1 = b"There are two types of cyptography: one that allows the Government to use brute force to break the code, and one that requires the Government to use brute force to break you."

for i in range(len(plain1)):
    key.append(plain1[i]^c[6][i])

key_string = ""
for i in key:
    key_string += chr(i)

print("key is:")
print(b2a_hex(key_string.encode()).decode())

print("\ncipher 8 is:")
for i in range(len(c[7])):
    print(chr(c[7][i]^key[i]),end="")

print()
```