A. Phân tích gợi ý

Ký tự dấu cách (space) trong mã ASCII thập phân là 32_{10} , dạng nhị phân 8-bit là 00100000_2 , dạng hex là 20_{16} . Ký tự trong khoảng [a-zA-Z] được biểu diễn dưới bảng sau:

Dec	Hex	Binary	Char	Char	Binary	Hex	Dec
65_{10}	41_{16}	01000001_2	A	a	01100001_2	61_{16}	97_{10}
66_{10}	42_{16}	01000010_2	B	b	01100010_2	62_{16}	98_{10}
67_{10}	43_{16}	01000011_2	C	c	01100011_2	63_{16}	99_{10}
68_{10}	44_{16}	01000100_2	D	d	01100100_2	64_{16}	100_{10}
69_{10}	45_{16}	01000101_2	E	e	01100101_2	65_{16}	101_{10}
70_{10}	46_{16}	01000110_2	F	f	01100110_2	66_{16}	102_{10}
71_{10}	47_{16}	01000111_2	G	g	01100111_2	67_{16}	103_{10}
72_{10}	48_{16}	01001000_2	H	h	01101000_2	68_{16}	104_{10}
73_{10}	49_{16}	01001001_2	I	i	01101001_2	69_{16}	105_{10}
74_{10}	$4A_{16}$	01001010_2	J	j	01101010_2	$6A_{16}$	106_{10}
75_{10}	$4B_{16}$	01001011_2	K	k	01101011_2	$6B_{16}$	107_{10}
76_{10}	$4C_{16}$	01001100_2	L	l	01101100_2	$6C_{16}$	108_{10}
77_{10}	$4D_{16}$	01001101_2	M	m	01101101_2	$6D_{16}$	109_{10}
78_{10}	$4E_{16}$	01001110_2	N	n	01101110_2	$6E_{16}$	110_{10}
79_{10}	$4F_{16}$	01001111_2	O	o	01101111_2	$6F_{16}$	111_{10}
80_{10}	50_{16}	01010000_2	P	p	01110000_2	70_{16}	112_{10}
81_{10}	51_{16}	01010001_2	Q	q	01110001_2	71_{16}	113_{10}
82_{10}	52_{16}	01010010_2	R	r	01110010_2	72_{16}	114_{10}
83_{10}	53_{16}	01010011_2	S	s	01110011_2	73_{16}	115_{10}
84_{10}	54_{16}	01010100_2	T	t	01110100_2	74_{16}	116_{10}
85_{10}	55_{16}	01010101_2	U	u	01110101_2	75_{16}	117_{10}
86_{10}	56_{16}	01010110_2	V	v	01110110_2	76_{16}	118_{10}
87_{10}	57_{16}	01010111_2	W	w	01110111_2	77_{16}	119_{10}
88_{10}	58_{16}	01011000_2	X	x	01111000_2	78_{16}	120_{10}
89_{10}	59_{16}	01011001_2	Y	y	01111001_2	79_{16}	121_{10}
90_{10}	$5A_{16}$	01011010_2	Z	z	01111010_2	$7A_{16}$	122_{10}

Khi XOR ký tự ‘space’ với ‘a’:

	00100000_2	(Space)
XOR	01100001_2	(a)
=	01000001_2	(A)

Khi XOR ký tự ‘space’ với ‘A’:

	00100000_2	(Space)
XOR	01000001_2	(A)
=	01100001_2	(a)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Điều đặc biệt ở đây là nếu chữ cái thường XOR với space sẽ được kết quả là chữ hoa của nó, và ngược lại nếu chữ cái hoa XOR với space lại được kết quả là chữ cái thường.

Một tính chất khác của XOR: **x XOR y XOR y = x.**

Giả sử có 2 bản rõ **p1** và **p2** mã hóa với cùng khóa **k.** Khi đó ta có 2 bản mã**: c1 = p1 ^ k** và **c2 = p2 ^ k**.

⇨ **c1 ^ c2 = (p1 ^ k) ^ (p2 ^ k) = p1 ^ p2**

Hướng giải mã của chúng ta như sau: lần lượt với các ký tự **c1[i]** và **c2[i]** ta tính **XOR** của chúng. Nếu kết quả là một chữ cái hoa trong alphabet thì trong **p1[i] và p2[i]** có thể sẽ có chữ cái đó dạng thường và ký tự space. Giả sử ta có ví dụ sau:

i	0	1	2	3
p1[i]	01000001_2 (A)	00100000_2 (space)	01100010_2 (b)	00100000_2 (space)
p2[i]	00100000_2 (space)	01111000_2 (x)	00100000_2 (space)	01011001_2 (Y)
k[i]	01001011_2 (K)	01101000_2 (h)	01101111_2 (o)	01100001_2 (a)

Tính **c1[i]** và **c2[i]:**

c1[i] = p1[i] ^ k	00001010_2	01001000_2	00001101_2	01000001_2
c2[i] = p2[i] ^ k	01101011_2	00010000_2	01001111_2	00111000_2
c1[i] ^ c2[i]	01100001_2 (a)	01011000_2 (X)	01000010_2 (B)	01111001_2 (y)

Từ ví dụ trên ta thấy **c1[i] XOR c2[i]** tiết lộ khá nhiều thông tin về bản rõ. Để có thể xác định đâu là ký tự space và đâu là ký tự alphabet, chúng ta sẽ XOR các bản mã với nhau để khôi phục các thông tin, kết hợp với dự đoán các từ có thể xuất hiện.

B. Tấn công One-Time Pad

Các bản mã bao gồm cả bản mã mục tiêu được chứa trong ciphertexts.txt, mỗi dòng một bản mã.

ciphertexts.txt | One-time pad.cpp

```
1  315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e
2  234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f
3  32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb
4  32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa
5  3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070
6  32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e6935226435 73b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4
7  32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce
8  315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3
9  271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027
10 466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83
11 32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904
```

Ghi các bản mã trên vào cipher_vector bằng đoạn code sau:

```
std::ifstream ciphertexts("One-time pad\\ciphertexts.txt");
if (!ciphertexts.is_open()){
    printf("Error ciphertexts\n");
    return 1;
}

// Get ciphertexts line by line from txt and write to vector
std::string line;
std::vector<std::string> cipher_vector;
while (std::getline(ciphertexts, line)){
    cipher_vector.push_back(line);
}
```

Để XOR các bản mã dạng hexa với nhau, cần chuyển các bản mã về dạng nhị phân và ngược lại. Ta dùng std::bitset để lưu trữ các bản mã dạng nhị phân:

```
// Constant: max bitset size
#define MAXSIZE 1500
// Create new names for std::bitset<N> types
typedef std::bitset<4> bit4_type;
typedef std::bitset<8> bit8_type;
typedef std::bitset<MAXSIZE> bit_max_type;
```

Các hàm để chuyển các bản mã từ hexa sang nhị phân và ngược lại:

```
bit_max_type hexToBit(const std::string &hex_text)
{
    // Insert space between every hex char. Example: "5a6b8c" -> "5 a 6 b 8 c"
    std::string hex(hex_text);
    for (int i = 1; i < hex.size(); i+=2){
        hex.insert(i, " ");
    }
    std::stringstream ss;
    ss << std::hex << hex;

    // Convert each hex char to 4-bit and copy to output stream
    std::ostringstream bit_stream;
    std::copy((std::istream_iterator<unsigned int>(ss)), std::istream_iterator<unsigned
int>(), std::ostream_iterator<bit4_type>(bit_stream));

    // bit_stream to bitset and return it
    bit_max_type hex_bitset(bit_stream.str());
    return hex_bitset;
}

std::string bitToHex(const std::string &bit){
    std::istringstream iss(bit);
    std::ostringstream oss;

    // Insert every 4-bit to input stringstream
    while (iss.good()){
        bit4_type bits;
        iss >> bits;
        oss << std::hex << bits.to_ullong();
    }
    std::string output = oss.str();
    // Remove last char '0'
    output.pop_back();
    return output;
}
```

Các hàm để XOR các bản mã dạng hexa:

```
bit_max_type bitXOR(const bit_max_type &first, const bit_max_type &second)
{
    return (first ^ second);
}

//      xxxxxxxxxxxxx
// XOR  xxxxx
// =    xxxxxxxxxxxxx
std::string hexXORHeadToHead(const std::string& hex1, const std::string& hex2)
{
    int hex1_length = hex1.length(), hex2_length = hex2.length();
    int diff = hex1_length - hex2_length;

    // If hex1 is longer than hex2 or equal to hex2
    if (diff >= 0)
    {
        // Convert hex strings to bitsets
        bit_max_type bit1 = hexToBit(hex1), bit2 = hexToBit(hex2);

        // If hex1 is longer than hex2
        //   bit1: xxxxxxxxxxxxxxxxxxx
        //   bit2:            xxxxxxxxx
        // ->bit2: xxxxxxxxx00000000000
        // Else don't change anything
        if (diff > 0)
        {
            bit2 <<= ((diff) * 4);
        }

        // XOR 2 bitsets
        bit_max_type bit1XORbit2 = bitXOR(bit1, bit2);

        // Remove '0'. Example: output: "0000000xxxxxxxx" -> "xxxxxxxx"
        std::string output = bitToHex(bit1XORbit2.to_string());
        int output_length = output.length();
        output.erase(output.begin(), output.begin() + output_length - hex1_length);
        return output;
    }
    else return hexXORHeadToHead(hex2, hex1);
}
```

Hàm để chuyển hex char sang dạng thập phân:

```
// Example: 'A' -> 10, 'F' -> 15
int hexValue(const unsigned char &hex_char)
{
    if ('0' <= hex_char && hex_char <= '9')
    {
        return hex_char - '0';
    }
    else if ('a' <= hex_char && hex_char <= 'f')
    {
        return hex_char - 'a' + 10;
    }
    else if ('A' <= hex_char && hex_char <= 'F')
    {
        return hex_char - 'A' + 10;
    }
    else throw std::invalid_argument("Invalid hex digit");
}
```

Cuối cùng là hàm dùng để chuyển hex string sang dạng ASCII string. Đối với những ký tự không nằm trong bảng chữ cái ta chuyển nó thành '*' để dễ phân biệt.

```
// hexToTextStar: convert hex string to ASCII string
// Replace non-alphabet char to '*'
std::string hexToTextStar(const std::string& input)
{
    const auto len = input.length();

    // If length is odd throw error
    if (len & 1) throw std::invalid_argument("Odd length");

    std::string output;
    output.reserve(len / 2);
    for (std::string::const_iterator p = input.begin(); p != input.end(); p++)
    {
        int c = hexValue(*p);
        p++;
        c = (c << 4) + hexValue(*p);

        output.push_back('|');

        if ((65 <= c && c <= 90) || (97 <= c && c <= 122))
        {
            output.push_back(c);
        }
        else
        {
            output.push_back('*');
        }
    }
    return output;
}
```

Dùng đoạn mã sau để XOR lần lượt 10 bản mã trước với bản mã cần giải mã và ghi kết quả ra file text:

```
// Pick up one ciphertext per time and XOR with the target (ciphertexts[10])
// Then write to cipherXORtarget file.
std::ofstream ciphersXORtarget("One-time pad\\cipherXORtarget.txt");

for (int i = 0; i < 10; ++i)
{

    std::string cipher_i_XOR_target = hexXORHeadToHead(cipher_vector[i], cipher_vector[10]);

    ciphersXORtarget << "cipher_vector[" << i << "] XOR target: "
                     << hexToTextStar(cipher_i_XOR_target) << std::endl;
}
```

Kết quả trong file ciphersXORtarget.txt ta được như sau:

```
[0]: **EC**C***T**S***EN**XE*HT******GQ*A****A*O********N**E*A*S*L**EF***O*O**ET***B**CT***W*YA****k*
[1]: ***E*E****DM******L*S*N***NT***N*O*****E**E****E*IC****RAU**R********N*O********T*NNEQ**Q*H******
[2]: ********E*H***S***U*SqE****QU**N*O****R***P******DE**V**NU**I**EAK**TM**EF*************z**A***a*
[3]: **********T***S***D***Dw**NAU******N******O*I*****I***E*O******EG*****R*I****T***E***Z*R******
[4]: *******U*TW***S**EB***Aw*******I**RAK***E**O*I*H**U****E*P***A***E*E*NM***A*************s*C****p*
[5]: ***R*E****TT**S*****SI********T******H***T***********R*I**V**E*S*E****T*E*A**R*R**A*O**C*y**R******a*
[6]: ***R*E****TT**S*****SI********O******Y*****E**A*I*****SN***Rg***R***N*E*OM*********EO***s**J*******o*
[7]: **EC**C********S***N*SMH***NT**I**I****R***A***H***AN****GU**TT******TM*********U***E*****A******
[8]: *HMP***********ZAG*N**CP***********EAS******M*C******ET***IRN***L*H******C****ET*********b**W*VA****w*
[9]: t**ES******S*E******D**YT****R*SA*W*W*S******N**P****T*E**RT**EA**EO*EYW****N*H*NRO***f**Y*R******
```

Nhìn vào cột [0], ta thấy có 9 ký tự '*' và 1 ký tự 't'. Vậy có khả năng ký tự đầu tiên [0] của bản rõ 9 là space và ký tự đầu tiên của bản rõ mục tiêu là 'T'.

Tương tự ở cột [1], ta thấy có 9 '*' và 1 'H'. Ký tự [1] của bản rõ 8 có thể là space và ký tự [1] của bản rõ mục tiêu là 'h'.

Ở cột [2], có 7 ký tự '*' và 2 ký tự 'E', 1 ký tự 'M'. Dựa vào 2 ký tự trước của mục tiêu là 'Th', có thể đây là 'The'.

Ở cột [3], có khá nhiều chữ cái khác nhau và '*'. Có thể dự đoán vị trí tiếp theo trong bản rõ mục tiêu là space: 'The '.

Tương tự ta có bảng dự đoán sau:

[i]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Char	**T**	**h**	**e**	**SP**	**s**	**e**	**c**	**u**	**e**	**t**	**SP**	**m**	**e**	**s**	**s**	**a**	**g**	**e**	**SP**	**i**	**s**	**SP**	**SP**	**W**

[i]	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Char	**h**	**t**	**n**	**SP**	**u**	**s**	**i**	**n**	**g**	**SP**	**a**	**SP**	**s**	**t**	**r**	**e**	**a**	**m**	**SP**	**c**	**i**	**p**	**h**	**e**

[i]	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
Char	**r**	**SP**	**SP**	**n**	**e**	**v**	**e**	**r**	**SP**	**u**	**s**	**e**	**SP**	**t**	**h**	**e**	**SP**	**k**	**e**	**y**	**SP**	**m**	**o**	**r**

[i]	72	73	74	75	76	77	78	79	80	81	82
Char	**e**	**SP**	**t**	**h**	**a**	**n**	**SP**	**o**	**n**	**c**	**e**

Do bản mã hexa có 166 ký tự nên bản rõ chỉ có 83 ký tự. Ta có bản rõ dự đoán như sau:

"The secuet message is Whtn using a stream cipher never use the key more than once"

Có thể sửa lại cho đúng chính tả như sau:

"The secret message is When using a stream cipher never use the key more than once"

Có tới 2 ký tự cách ở sau "is" và "cipher". Để hoàn thiện nốt bản rõ, XOR bản rõ tạm thời với bản mã để tìm khóa:

```
66396e89c9dbd8cc9874352acd6395102eafce78aa65ed28a07f6bc98d29c50b69b0339a19f8aa401a9c6d708f80c0
66c76ffef0123148cdd8e802d05ba98777335daefcecd59c433a6b268b60bf4ef03c9a61
```

XOR khóa này với các bản mã khác ta được:

```
[0]: We can factor the numxer 15 with quantum computer□. We can also factor the number 1%□
□□Ya□□□>
[1]: Euler would probably □njoy that now his theorem bicomes a corner stone of crypto - Q□
[2]: The nice thing about Qeeyloq is now we cryptograpders can drive a lot of fancy cars0□
[3]: The ciphertext produc□d by a weak encryption algo~ithm looks as good as ciphertext `□
□□D~,v□=0□□.L□□□1Z□□□"□□cj□
[4]: You don't want to buy:a set of car keys from a guu who specializes in stealing cars0□
[5]: There are two types o| cryptography - that which {ill keep secrets safe from your ly□
[6]: There are two types o| cyptography: one that allo{s the Government to use brute fors□
□□Yi4w~"u□□□.@X*□ge□□□"□□"p□□s□□□2□o□□□□□G<□□□□□:Sl□□□?□□□□□□□f□£□
[7]: We can see the point mhere the chip is unhappy if,a wrong bit is sent and consumes }□
[8]: A (private-key)  encrcption scheme states 3 algorethms, namely a procedure for geneb□
[9]:  The Concise OxfordDiytionary (2006) defines cry|to as the art of  writing o r solf□□
[10]: The secret message is  When using a stream cipher  never use the key more than once
```

Sửa bản rõ [1] như sau: **“Euler would probably enjoy that now his theorem becomes a corner stone of crypto – “**. XOR bản rõ trên với bản mã sẽ được khóa mới như sau:

```
66396e89c9dbd8cc9874352acd6395102eafce78aa7fed28a07f6bc98d29c50b69b0339a19f8aa401a9c6
d708f80c066c763fef0123148cdd8e802d05ba98777335daefcecd59c433a6b268b60bf4ef03c9a6151f6
d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f
```

Các bản rõ thu được sau khi XOR với khóa mới:

```
[0]: We can factor the number 15 with quantum computers. We can also factor the number 1tN□□□□M□U□O□NT7□□□□CST□H□□ □M□  V   ��
□�Ya□□�>
[1]: Euler would probably enjoy that now his theorem becomes a corner stone of crypto -
[2]: The nice thing about Keeyloq is now we cryptographers can drive a lot of fancy carsaCN+□□M-□□E□[□��n���$□□'��5o
[3]: The ciphertext produced by a weak encryption algorithm looks as good as ciphertext 1□□□□□□□U□YO□ 6□□
□@SE□□□□□□□□□□
��D~,v□=0��.L□□□1Z□□□"��cj□
[4]: You don't want to buy a set of car keys from a guy who specializes in stealing carsaCN"□□□O'□T
 B □□E□H□M□□□□□□M□□□/��[| p
[5]: There are two types of cryptography - that which will keep secrets safe from your l(□□□□Y□□□□E□B $□□E□O□TT□
□□
M□□H ��Ni5"d5s���5□□□□t □□□*��mq��a□□□"□w□窈□□□+□□□□□5]%□□
[6]: There are two types of cyptography: one that allows the Government to use brute for"□N□□Y□□□□KO□H U□
□B_ □□□O□□□\□L
□□Yi4w~"u□□□.@X*□ge□□□"□□"p□□s□□□2□o□□□□□G<□□□□□:Sl□□□?□□□□□□□f□£□
[7]: We can see the point where the chip is unhappy if a wrong bit is sent and consumes ,□□
N   □□□□    □O(U□
□□□N□□□ □□□□□□A□□OeeQ□1}□□
[8]: A (private-key)  encryption scheme states 3 algorithms, namely a procedure for gene3□□□ □M□□
SCNAe□□
□B□U□
E   □□M□□G□□□_e+e;pq□Ś'□□□□re□□□"□□mv□c□□□ □s□□□'
[9]:  The Concise OxfordDictionary (2006) defines crypto as the art of  writing o r sol7□ □N□□□□ □O[□□□n□□□$□□'□5o
[10]: The secret message is: When using a stream cipher, never use the key more than onceQ□□Q□H□□□□$□[□□□n□□□$□□'□5o
```

Đến đây bản mã cần tìm đã đầy đủ:

```
“The secret message is: When using a stream cipher, never
use the key more than once”
```