How do I solve the OTP?

The One-Time Pad (OTP) is secure only if used once. However, since the same key was reused across all ciphertexts, the encryption becomes vulnerable and easy to decrypt. We can use XOR pairs of ciphertexts to reveal information about the plaintexts.
XOR of two ciphertexts formula is as follows:

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$$

The key cancels out, leaving only the XOR of two plaintexts. If one of the plaintexts is known, the key can be revealed and hence used to decrypt all messages.

Given that information, I want to convert Hexadecimal ciphertexts to binary. This will give binary codes. From the assignment, this is a binary representation of the code. Then, XOR ciphertexts to obtain the XOR of plaintexts.

Ciphertext 1:
BB3A65F6F0034FA957F6A767699CE7FABA855AFB4F2B520AEAD612944A801E
Binary: 10111011 00111010 01100101 11110110 11110000 00000011 01001111 10101001 01010111 11110110 10100111 01100111 01101001 10011100 11100111 11111010 10111010 10000101 01011010 11111011 01001111 00101011 01010010 00001010 11101010 11010110 00010010 10010100 01001010 10000000 00011110
Ciphertext 2:
BA7F24F2A35357A05CB8A16762C5A6AAAC924AE6447F0608A3D11388569A1E
Binary: 10111010 01111111 00100100 11110010 10100011 01010011 01010111 10100000 01011100 10111000 10100001 01100111 01100010 11000101 10100110 10101010 10101100 10010010 01001010 11100110 01000100 01111111 00000110 00001000 10100011 11010001 00010011 10001000 01010110 10011010 00011110
Ciphertext 3:
A67261BBB30651BA5CF6BA297ED0E7B4E9894AA95E300247F0C0028F409A1E
Binary: 10100110 01110010 01100001 10111011 10110011 00000110 01010001 10111010 01011100 11110110 10111010 00101001 01111110 11010000 11100111 10110100 11101001 10001001 01001010 10101001 01011110 00110000 00000010 01000111 11110000 11000000 00000010 10001111 01000000 10011010 00011110
Ciphertext 4:
A57261F5F0004BA74CF4AA2979D9A6B7AC854DA95E305203EC8515954C9D0F

Binary: 10100101 01110010 01100001 11110101 11110000 00000000 01001011 10100111 01001100 11110100 10101010 00101001 01111001 11011001 10100110 10110111 10101100 10000101 01001101 10101001 01011110 00110000 01010010 00000011 11101100 10000101 00010101 10010101 01001100 10011101 00001111

Ciphertext 5:

BB3A70F3B91D48E84DF0AB702ECFEEB5BC8C5DA94C301E0BECD241954C831E

Binary: 10111011 00111010 01110000 11110011 10111001 00011101 01001000 11101000 01001101 11110000 10101011 01110000 00101110 11001111 11101110 10110101 10111100 10001100 01011101 10101001 01001100 00110000 00011110 00001011 11101100 11010010 01000001 10010101 01001100 10000011 00011110

Ciphertext 6:

A6726DE8F01A50E849EDBC6C7C9CF2B2A88E19FD423E0647ECCB04DD4C9D1E

Binary: 10100110 01110010 01101101 11101000 11110000 00011010 01010000 11101000 01001001 11101101 10111100 01101100 01111100 10011100 11110010 10110010 10101000 10001110 00011001 11111101 01000010 00111110 00000110 01000111 11101100 11001011 00000100 11011101 01001100 10011101 00011110

Ciphertext 7:

BC7570BBBF1D46E85AF9AA6C7A9CEFA9E9825CFD5E3A0047F7CD009305A71E

Binary: 10111100 01110101 01110000 10111011 10111111 00011101 01000110 11101000 01011010 11111001 10101010 01101100 01111010 10011100 11101111 10101001 11101001 10000010 01011100 11111101 01011110 00111010 00000000 01000111 11110111 11001101 00000000 10010011 00000101 10100111 00011110

Then, to recover the key, I need $K=C \oplus P$. Once we have enough key bytes, we can decrypt all messages. Since the key is reused, we can manually refine it by: Identifying common English phrases. Adjusting incorrect words by updating the key.

The provided Python code automates: Hex decoding XOR operations Key recovery using known plaintext Decryption using the recovered key.