**Problem 1: Understanding One-Time Pad**

A one-time pad­refers to a cryptographic system where a secret key is generated from a fully, or uniformly, random set of values and used only once to both encrypt and decrypt a single message. If the following conditions are met, then use of a one-time pad enables “perfect secrecy”, which indicates that given the ciphertext of the message, an adversary gains no additional information about the underlying plaintext message, even if the adversary has theoretically infinite computing power:

* The Secret Key is only used once
* The values used to generate the key must be truly random
* The secret key is securely shared between the sending and receiving parties
* The secret key must be at least as long as the plaintext being encrypted

This last requirement is based on a theorem by Claude Shannon, which is stated formally as:

*Any cipher achieving perfect secrecy requires that |K| ≥ |M |*, where K is the key space and M is the plaintext message space. Like other encryption systems, one-time pads also adhere to Kerchoff’s Principle, which states: [The method of encryption] *must not be required to be secret, and it must be able to fall into the enemy’s hands without causing inconvenience.* This principle implies that even if an adversary has knowledge of the encryption technique they will be unable to decrypt without the secret key.

However, the requirement of a key being at least as long as the plaintext message limits its practical use in many modern applications, including anything related to digital media.

Sources:

<https://www.techtarget.com/searchsecurity/definition/one-time-pad>

<https://courses.grainger.illinois.edu/CS407/fa2023/Scribe%202.pdf>

**Problem 2: One Time Pad Implementation**

# **Alice's Encryption**

One-time pad encryption means that Alice starts with a plaintext message typed by the user. The message is then converted into digital form, typically in bytes. Next, Alice’s program generates a random key that’s exactly as long as the message. This random key is needed for security, because if the key is truly unpredictable, it prevents anyone from guessing the original text. Although python cannot generate a true random value, Alice can use the python “secrets” module to mimic it with a pseudorandom value for the key. The encryption step uses the XOR (exclusive OR) operation, which compares each bit of the plaintext with the corresponding bit of the key. If the bits are different, it writes a 1; if they are the same, it writes a 0. This produces the ciphertext, which is unreadable unless you have the key. Finally, Alice saves the ciphertext and key in a shared or secure location. If the key stays secret and is never reused, the one-time pad offers nearly perfect security. This approach makes sure that without the key, decryption is impossible.

# **Bob's Decryption**

While Alice is responsible for encrypting a message with the one-time pad, Bob’s job is to do the reverse process: decryption. He starts by receiving the ciphertext and the key, which are stored here in separate files. He reads them in, converting from hexadecimal format back into bytes, making sure he works with the exact same data used by Alice. Then, Bob performs the XOR operation again, this time using the ciphertext and the key as inputs. Because XOR is its own inverse, applying it once more recovers the original plaintext. If Alice’s key and ciphertext were transmitted correctly, Bob’s decrypted output will match the exact message Alice sent. After that, Bob simply converts the plaintext bytes back into letters, displaying them on his screen. This part of the process shows the importance to keep the key safe, since anyone who has it can completely read the secret messages.

**Problem 3: Implementing Many-Time Pad**

**Problem 4: Cryptanalysis of Many-Time Pad**

The fundamental weakness introduced in the reuse of a key to encrypt multiple texts, or the many-time pad, is that the ciphertext is no longer truly random and patterns may be discoverable by comparing multiple texts, which we assume the eavesdropper can obtain.

The first step in a computational approach to cryptanalysis of a many-time pad is to XOR the available ciphertexts with each other. Each pair of XORed ciphertexts will yield the XOR of the the two corresponding plaintexts. These XORed plaintexts are then potentially vulnerable to frequency analysis and dictionary attacks, but our approach employed XORing the plaintext pairs with a suspected or known plaintext word, or crib dragging.

Our attached program first sanitizes the ciphertexts to make sure the characters are valid hexidecimal, and then make sure that the ciphertexts are even (since letters and numbers are represented by two hex characters) It then creates a placeholder key value of null bytes the same length as the target ciphertext. The program then provides two options. The first a takes user input for a crib value and a starting index in the target ciphertext to XOR the crib value with. The key value is then updated with the resulting characters. The second option is to test the existing key value by encrypting the target text with it, and then checking to see if the key is correct be re-encrypting with the existing key value and comparing with the original ciphertext.

Additionally, we include a code to validate the key is complete and valid for the entire target ciphertext by using the same checking mechanism – decrypting the message and reencrypting by XORing with the now complete key and comparing the re-encrypted ciphertext with the original ciphertext. If the two match, the key is complete.