International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 03 Issue: 06 | June - 2019 SJIF Rating: 5.713 ISSN: 2582-3930

Unique Sequence Generator for Symmetric Encryption and Decryption

Dr. G. Sreedhar
Department of Computer Science
Rashtriya Sanskrit Vidyapeetha, Tirupati

Abstract This paper introduces a novel symmetric encryption and decryption method utilizing a pseudo-random number generator (Xorshift) to produce a unique sequence. This sequence is employed to shift characters in a plaintext message, achieving encryption, and can be used to restore the original message during decryption. The approach is implemented through a web-based application using HTML and JavaScript.

***Key Words*:** Pseudo-random number generator, encryption, decryption,

1.INTRODUCTION

Cryptography plays a pivotal role in securing data across various platforms. Traditional encryption algorithms often rely on complex mathematical functions. This study explores an alternative method by generating a unique sequence using a pseudo-random number generator and employing it for both encryption and decryption processes.

2. METHODOLOGY

2.1 Pseudo-Random Number Generation

A Pseudo-Random Number Generator (PRNG) is an algorithm that produces a sequence of numbers that approximates the properties of random numbers. Unlike true random number generators, which rely on physical processes, PRNGs are deterministic and produce the same sequence from a given initial state, known as the **seed**.

The **Xorshift** algorithm, introduced by George Marsaglia in 2003, is a class of PRNGs that operates by repeatedly applying the **exclusive OR (XOR)** operation combined with bit shifts on an internal state variable. This method is computationally efficient and produces sequences with good statistical properties, making it suitable for applications like encryption where speed and randomness are crucial.

The basic Xorshift algorithm involves the following steps:

1. **State Initialization**: Start with a non-zero seed value.
2. **Bit Shifting and XOR Operations**: Apply a series of bit shifts and XOR operations to the state variable.
3. **Output Generation**: The resulting value after the operations is the next number in the sequence.
4. **State Update**: The state is updated for the next iteration.

This process is repeated to generate a sequence of pseudo-random numbers.

2.2 Sequence Generation

The generated sequence consists of 26 unique numbers corresponding to the 26 letters of the English alphabet. This sequence is used to shift the position of each character in the plaintext message.

2.3 Encryption Process

2.3.1 Overview

In symmetric encryption, the same key is used for both encryption and decryption. In this project, the "key" is a unique sequence generated using the Xorshift pseudo-random number generator. This sequence determines how each character in the plaintext is transformed into ciphertext.

2.3.2 Detailed Steps

1. **Input and Sequence Generation**: The user provides a seed value. The Xorshift algorithm uses this seed to generate a sequence of 26 unique numbers, each corresponding to a letter in the English alphabet.
2. **Mapping Letters to Sequence**: The 26 letters of the English alphabet are assigned to the generated sequence. For example, if the sequence starts as [3, 1, 4, 2, 5, ...], then 'A' maps to 3, 'B' to 1, 'C' to 4, and so on.
3. **Encrypting the Plaintext**:
    - For each character in the plaintext:
        - If the character is an uppercase letter (A-Z):
            - Determine its position in the alphabet (e.g., 'A' = 0, 'B' = 1, ..., 'Z' = 25).
            - Add the corresponding value from the sequence to this position.
            - Use modulo 26 arithmetic to ensure the result wraps around within the alphabet range.
            - Replace the original character with the new character obtained from the shifted position.
        - If the character is not an uppercase letter, it remains unchanged.
4. **Output**: The transformed characters are concatenated to form the ciphertext.

Example:

- **Seed**: 12345
- **Generated Sequence**: [3, 1, 4, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
- **Plaintext**: "HELLO"
- **Ciphertext**: "KHOOR"

2.4 Decryption Process

The ciphertext is processed by shifting each character backward by the corresponding value in the sequence, restoring the original plaintext.

2.4.1 Overview

Decryption is the reverse process of encryption. Since the same sequence is used for both operations, applying the inverse transformation restores the original plaintext.

2.4.2 Detailed Steps

1. **Input and Sequence Generation**: The user provides the same seed value used during encryption. The Xorshift algorithm generates the same sequence of 26 unique numbers.
2. **Mapping Letters to Sequence**: The 26 letters of the English alphabet are assigned to the generated sequence, identical to the encryption process.
3. **Decrypting the Ciphertext**:
    - For each character in the ciphertext:
        - If the character is an uppercase letter (A-Z):
            - Determine its position in the alphabet (e.g., 'A' = 0, 'B' = 1, ..., 'Z' = 25).
            - Subtract the corresponding value from the sequence from this position.
            - Use modulo 26 arithmetic to ensure the result wraps around within the alphabet range.
            - Replace the encrypted character with the new character obtained from the shifted position.
        - If the character is not an uppercase letter, it remains unchanged.
4. **Output**: The transformed characters are concatenated to form the decrypted plaintext.

Example:

- **Seed**: 12345
- **Generated Sequence**: [3, 1, 4, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
- **Ciphertext**: "KHOOR"
- **Decrypted Plaintext**: "HELLO"

In this example, each letter of "KHOOR" is shifted backward by the corresponding value in the sequence, restoring the original plaintext "HELLO".

3. ALGORITHM

The following pseudocode outlines the steps involved in generating the unique sequence, encrypting the message, and decrypting the ciphertext.

3.1 Generate Unique Sequence

```
Function GenerateUniqueSequence(seed):
    Initialize empty list sequence
    Initialize empty set seen
    Initialize random number generator rng using seed
    While length of sequence < 26:
        num = Floor(rng() * 26) + 1
        If num is not in seen:
            Add num to seen
            Append num to sequence
    Return sequence
```

3.2 Encrypt Message

```
Function EncryptMessage(message, sequence):
    Initialize alphabet as "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Initialize empty string encrypted_message

    For each character in message:
        If character is in alphabet:
            index = Index of character in alphabet
            shift = sequence[index]
            new_index = (index + shift) mod 26
            Append alphabet[new_index] to encrypted_message
        Else:
            Append character to encrypted_message

    Return encrypted_message
```

3.3 Decrypt Message

```
Function DecryptMessage(encrypted_message, sequence):
    Initialize alphabet as "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Initialize empty string decrypted_message
    For each character in encrypted_message:
        If character is in alphabet:
            index = Index of character in alphabet
            shift = sequence[index]
            new_index = (index - shift + 26) mod 26
            Append alphabet[new_index] to decrypted_message
        Else:
            Append character to decrypted_message

    Return decrypted_message
```

4. IMPLEMENTATION

The implementation is carried out using HTML for the user interface and JavaScript for the logic. The user inputs a seed and a message, and the application displays the generated sequence, encrypted message, and decrypted message.

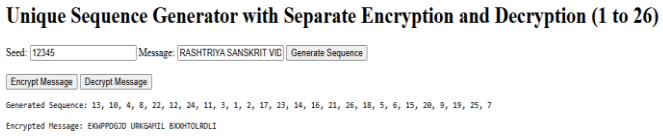

Figure 1: Encryption using sequence generator

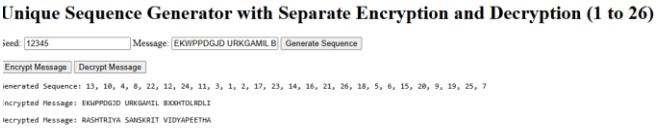

Figure 2: Decryption using sequence generator

5. CONCLUSION

This approach provides a simple yet effective method for symmetric encryption and decryption. The use of a pseudo-random number generator ensures that the sequence is unique and reproducible, allowing for secure communication. However, the security of this method depends on the unpredictability of the pseudo-random number generator and the secrecy of the seed. The unique sequence generator offers a straightforward method for encrypting and decrypting messages. While not suitable for high-security applications, it serves as an educational tool for understanding the principles of symmetric encryption.

6. REFERENCES

1. Marsaglia, G. (2003). Xorshift Random Number Generators. *Journal of Statistical Software*, 14(6), 1–10. https://www.jstatsoft.org/article/view/v014i06
2. Rivest, R. (1987). RC2: A Fast Block Cipher. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1–9. https://ieeexplore.ieee.org/document/623380
3. Schneier, B., Kelsey, J., & Wagner, D. (1997). Digital Cell Phone Crypto Cracked. *Wired*. https://www.wired.com/1997/03/digital-cell-phone-crypto-cracked
4. Lai, X., & Massey, J. (1991). International Data Encryption Algorithm. *Proceedings of the 1991 IEEE International Conference on Communications*, 1, 131–136. https://ieeexplore.ieee.org/document/140016
5. Wikramaratna, R. S. (1989). ACORN — A New Method for Generating Sequences of Uniformly Distributed Pseudo-random Numbers. *Journal of Computational Physics*, 83(1), 16–31. https://doi.org/10.1016/0021-9991(89)90002-2
6. Jenkins, R. J. (1996). ISAAC: A Fast Cryptographic Pseudorandom Number Generator. *Fast Software Encryption*, 41–49. https://www.schneier.com/academic/archives/1996/02/isaac_a_fast_cr.html
7. Bhattacharjee, K., Maity, K., & Das, S. (2018). A Search for Good Pseudo-random Number Generators: Survey and Empirical Studies. *arXiv*. https://arxiv.org/abs/1811.04035
8. 8. Lemire, D., & O'Neill, M. E. (2014). PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. *Proceedings of the 2014 ACM SIGPLAN International Conference on Functional Programming*, 1–12. https://doi.org/10.1145/2628136.2628148