Abstract

In the modern digital landscape, securing data transmission has become paramount as cyber threats evolve in complexity. This report explores a lightweight cryptographic algorithm utilizing a variable-length key for encrypting and decrypting text. The key is generated dynamically using random printable ASCII characters, ensuring unique encryption for each session. The encryption process involves converting text to ASCII values, applying modular arithmetic with the key, and generating a ciphertext that is unreadable without the correct key. The decryption process reverses these transformations to retrieve the original plaintext.

This method is efficient, with linear time complexity, making it suitable for real-time applications and resource-constrained environments like IoT devices. Despite its simplicity, it provides a robust foundation for understanding cryptographic principles, offering practical applications in secure messaging and data protection. The report evaluates the algorithm's effectiveness, highlights its advantages and limitations, and discusses potential enhancements for broader adoption.

Contents

Sl. No. Topic **Page no.**

Introduction

In today's interconnected world, the need for securing sensitive information has never been greater. Every second, vast amounts of data are transmitted over networks, making it vulnerable to unauthorized access and cyberattacks. Cryptography, the science of encoding and decoding information, plays a pivotal role in safeguarding this data by preventing interception and misuse.

This report introduces a lightweight cryptographic algorithm based on variable-length keys. The algorithm dynamically generates random keys using printable ASCII characters, ensuring unique encryption for each session. By leveraging simple mathematical operations, this method transforms plaintext into ciphertext that is incomprehensible without the correct key. Unlike complex cryptographic systems, this approach prioritizes simplicity and efficiency, making it accessible for educational purposes and practical use in constrained environments.

The purpose of this report is to analyze the design, implementation, and performance of the variable-length key cryptographic algorithm. It aims to demonstrate its effectiveness in securing textual data, evaluate its strengths and limitations, and explore its potential applications in real-world scenarios.

Objectives

1. **To demonstrate the functionality of variable-length key encryption**:
   Showcase the process of encrypting and decrypting textual data using dynamically generated keys.
2. **To analyze the security of the encryption algorithm**:
   Evaluate how the use of variable-length random keys enhances the confidentiality of data and resists brute-force attacks.
3. **To study the computational efficiency of the algorithm**:
   Assess the algorithm's linear time complexity and its suitability for real-time applications.
4. **To understand the practicality of key-based encryption**:
   Explore the challenges and solutions associated with symmetric key dependency, including secure key sharing.
5. **To explore potential applications**:
   Investigate scenarios where the algorithm can be applied, such as secure messaging, lightweight IoT encryption, and educational tools for learning cryptographic principles.

Methodology

The cryptographic algorithm is designed to encrypt and decrypt textual data using a variable-length secret key. The key is generated randomly using printable ASCII characters, ensuring a unique encryption for each session. Below are the detailed steps for both the encryption and decryption processes.

Encryption Process

1. **Key Generation**:
    - A secret key is dynamically generated using random printable ASCII characters. The length of the key is specified by the user. This key is used for both encryption and decryption.
2. **Convert Text to ASCII**:
    - The input plaintext is converted to a list of ASCII values, where each character is represented by its corresponding numerical value. This allows for mathematical manipulation of the text.
3. **Apply Secret Key**:
    - The ASCII values of the plaintext characters are modified by adding the ASCII value of the corresponding character from the secret key. The key is applied in a cyclic manner, ensuring the encryption process uses all characters of the key.
4. **Modular Arithmetic**:
    - After applying the key, a modulo operation is performed on the modified ASCII values (mod 256), ensuring that the encrypted values remain within the valid ASCII range (0–255).
5. **Generate Ciphertext**:
    - The modified ASCII values are then converted back to characters, resulting in the ciphertext. This final encrypted text is transmitted to the receiver.

Decryption Process

1. **Convert Ciphertext to ASCII**:
    - The ciphertext is first converted to ASCII values to prepare for the decryption process.
2. **Apply Secret Key in Reverse**:
    - The secret key is applied to the ASCII values of the ciphertext, but this time, the ASCII value of the key characters is subtracted from the ciphertext values.
3. **Modular Arithmetic**:
    - A modulo operation (mod 256) is applied to the modified ASCII values, reversing the encryption process.
4. **Generate Plaintext**:
    - The decrypted ASCII values are converted back to characters to reconstruct the original plaintext.
5. **Final Output**:
    - The original message is restored, demonstrating the effectiveness of the encryption and decryption processes.

This methodology ensures that the encryption algorithm is both simple and efficient, providing an accessible introduction to cryptographic principles. It is also adaptable for real-time applications due to its linear time complexity.

Applications

The variable-length key cryptographic algorithm offers a range of practical applications due to its simplicity, efficiency, and adaptability. Below are some key areas where this algorithm can be effectively utilized:

1. **Secure Messaging**:
    - The algorithm can be applied in messaging applications to encrypt text messages exchanged over unsecured channels, ensuring that only the intended recipient with the correct key can decrypt and read the message.

2. **Cloud Data Security**:
    - Before uploading sensitive files to cloud storage, they can be encrypted using this algorithm. This protects the data from unauthorized access, ensuring its confidentiality even in the event of cloud service breaches.
3. **IoT and Edge Devices**:
    - IoT devices, which often operate with limited resources, can benefit from this lightweight encryption algorithm. It provides a secure means to protect data exchanged between IoT devices and networks without consuming significant computational power.
4. **Embedded Systems**:
    - In embedded systems, where computational resources may be constrained, this algorithm offers a simple yet effective solution for encrypting sensitive information, such as personal data or system configurations.
5. **Educational Tools for Cryptography**:
    - Due to its simplicity, the algorithm is ideal for use in educational environments where students can learn basic cryptographic techniques. It helps demonstrate key concepts like encryption, decryption, and key management in an easily understandable manner.
6. **Wireless Communication**:
    - The algorithm can be employed in wireless communication systems, such as Bluetooth or Zigbee, to secure data transmission between devices, ensuring that only authorized devices can access the transmitted data.
7. **Data Protection in Small-Scale Applications**:
    - For small-scale applications where computational power and memory are limited, this algorithm provides an efficient means of encrypting and protecting sensitive data without the need for complex cryptographic systems.

CODE

```
import random

def generate_variable_length_key(length):
    """Generate a variable-length key (string of random characters)."""
    return ''.join([chr(random.randint(33, 126)) for _ in range(length)])  # Random printable ASCII characters

def encrypt_with_variable_key(text, secret_key):
    """Encrypt the given text using the provided variable-length key."""
    try:
        key_len = len(secret_key)
        padded_ascii = [ord(char) for char in text]

        # Step 1: Encrypt using key characters
        for i in range(len(padded_ascii)):
            key_char = ord(secret_key[i % key_len])  # Rotate over the key
            padded_ascii[i] = (padded_ascii[i] + key_char) % 256  # Encrypt using key characters

        # Step 2: Convert ASCII to characters and create ciphertext
        cipher_text = ''.join(chr(val) for val in padded_ascii)
        return cipher_text
    except Exception as e:
        print(f"Encryption error: {e}")
        return None

def decrypt_with_variable_key(cipher_text, secret_key):
    """Decrypt the given cipher text using the provided variable-length key."""
    try:
        key_len = len(secret_key)
        encrypted_ascii = [ord(char) for char in cipher_text]

        # Step 1: Decrypt using key characters
        for i in range(len(encrypted_ascii)):
            key_char = ord(secret_key[i % key_len])
            encrypted_ascii[i] = (encrypted_ascii[i] - key_char) % 256  # Decrypt using key characters

        # Step 2: Convert ASCII values back to characters
        plain_text = ''.join(chr(val) for val in encrypted_ascii)
        return plain_text
    except Exception as e:
        print(f"Decryption error: {e}")
        return None

# Example usage
if __name__ == "__main__":
    # Original text (including symbols and non-standard characters)
    text = input("Enter your message: ")
    print(f"Original Text: {text}")

    # Generate a variable-length secret key
    key_length = int(input("Enter the length of the secret key: "))
    secret_key = generate_variable_length_key(key_length)
    print(f"Secret Key: {secret_key}")

    # Encrypt the text
    cipher_text = encrypt_with_variable_key(text, secret_key)
    print(f"Cipher Text: {cipher_text}")

    # Decrypt the text
    decrypted_text = decrypt_with_variable_key(cipher_text, secret_key)
    print(f"Decrypted Text: {decrypted_text}")
```

Results –

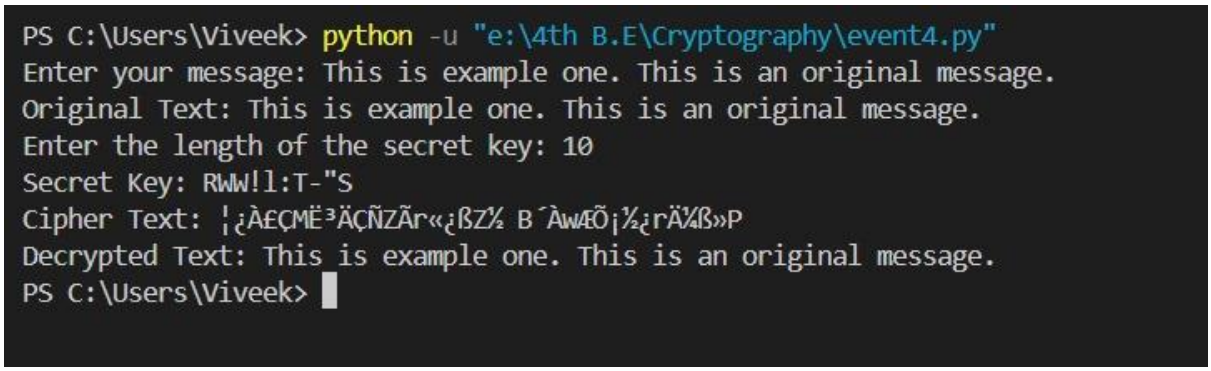

Fig1. Example 1

The above example depicts the encryption and decryption of a message line. Here, the length of the secret key is kept as a 2-digit number. Encryption and decryption, as can be seen here, are both successful.

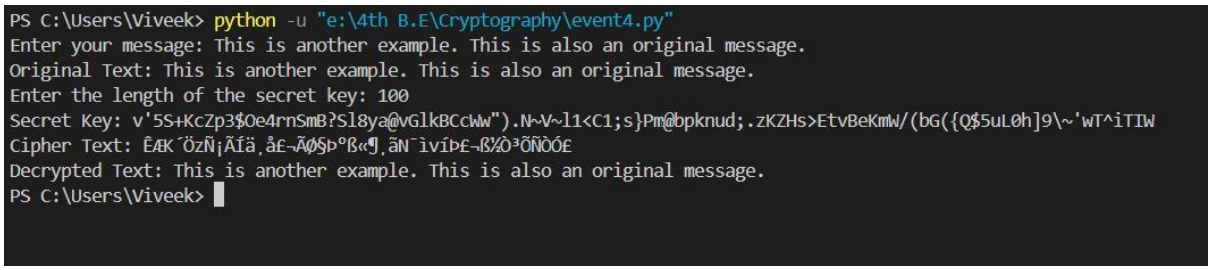

Fig2. Example 2

The above example depicts the encryption and decryption of another message line. Here, the length of the secret key is kept as a 3-digit number. Encryption and decryption, as can be seen here, are both successful.

Conclusion

In an era where digital communication and data sharing are integral to everyday life, ensuring the security of sensitive information is crucial. The cryptographic algorithm presented in this report, which uses a dynamically generated variable-length key, offers a simple yet effective approach to encrypting and decrypting data. By employing basic mathematical operations such as ASCII conversion and modular arithmetic, this algorithm provides a reliable method for protecting textual data from unauthorized access.

Despite its simplicity, the algorithm ensures that each encryption session uses a unique key, significantly enhancing security against common attacks like brute-force decryption. Its linear time complexity makes it an efficient choice for real-time applications, while its lightweight nature allows it to be implemented on devices with limited computational resources, such as IoT and embedded systems.

Through the analysis and implementation presented, this cryptographic technique can be effectively applied to various scenarios, including secure messaging, cloud data protection, and securing IoT devices. While there are limitations such as the simplicity of the key structure and the need for secure key distribution, the algorithm provides an accessible foundation for understanding cryptographic principles and offers a practical solution for many applications.

In conclusion, the proposed encryption method offers a robust, efficient, and educational approach to securing digital data. Further research and optimization could enhance its applicability, particularly for larger-scale or more complex systems, making it a valuable tool in the ever-evolving field of cybersecurity.

References

1. **Cryptography Basics - GeeksforGeeks**
   https://www.geeksforgeeks.org/cryptography/
2. **Understanding Cryptography - SpringerLink**
   https://link.springer.com/book/9783642041006
3. **Introduction to Cryptography - TutorialsPoint**
   https://www.tutorialspoint.com/cryptography/
4. **Cryptographic Algorithms - Wikipedia**
   https://en.wikipedia.org/wiki/Cryptographic_algorithm
5. **What is Cryptography? - IBM Security**
   https://www.ibm.com/topics/cryptography
6. **Introduction to Cryptography with Python - DigitalOcean**
   https://www.digitalocean.com/community/tutorials/
7. **The Basics of Encryption - Stanford University**
   https://crypto.stanford.edu/~dabo/cs255/
8. **Cryptography Overview - Microsoft Docs**
   https://docs.microsoft.com/en-us/dotnet/standard/security/cryptography
9. **How Cryptography Works - Techradar**
   https://www.techradar.com/how-to/how-cryptography-works
10. **Modern Cryptography Techniques - Coursera**
    https://www.coursera.org/learn/cryptography