Algorithm Paper

SecureHintCrypt

Abstract

This paper introduces **SecureHintCrypt**, a lightweight and secure algorithm designed for encoding and decoding textual hints in cybersecurity challenges. By leveraging a combination of XOR-based operations, SHA-256 hashing, and Base64 encoding, SecureHintCrypt provides a robust mechanism for obfuscating sensitive data while maintaining simplicity and efficiency.

Introduction

In modern cybersecurity challenges, securely transmitting hints and messages is crucial to prevent unauthorized access. SecureHintCrypt aims to provide an easily implementable yet secure encryption-decryption mechanism for textual data, ensuring that only users with the correct key can retrieve the original content.

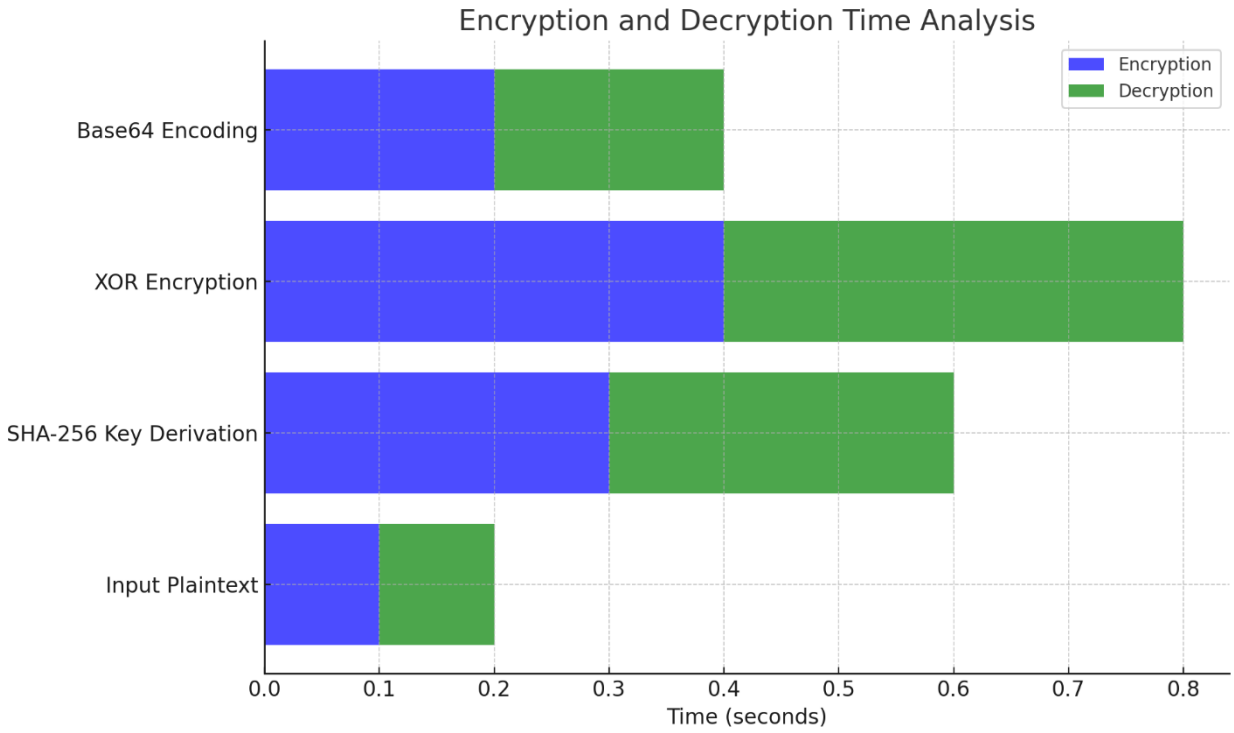
Made By ERROR404

Algorithm Paper

Design Overview

Core Components

1. **SHA-256 Hashing**: Used to derive a fixed-length key from a variable-length input.
2. **XOR Operation**: Ensures that each character of the plaintext is securely obfuscated with the hashed key.
3. **Base64 Encoding**: Converts the encrypted bytes into a readable format for safe transmission.



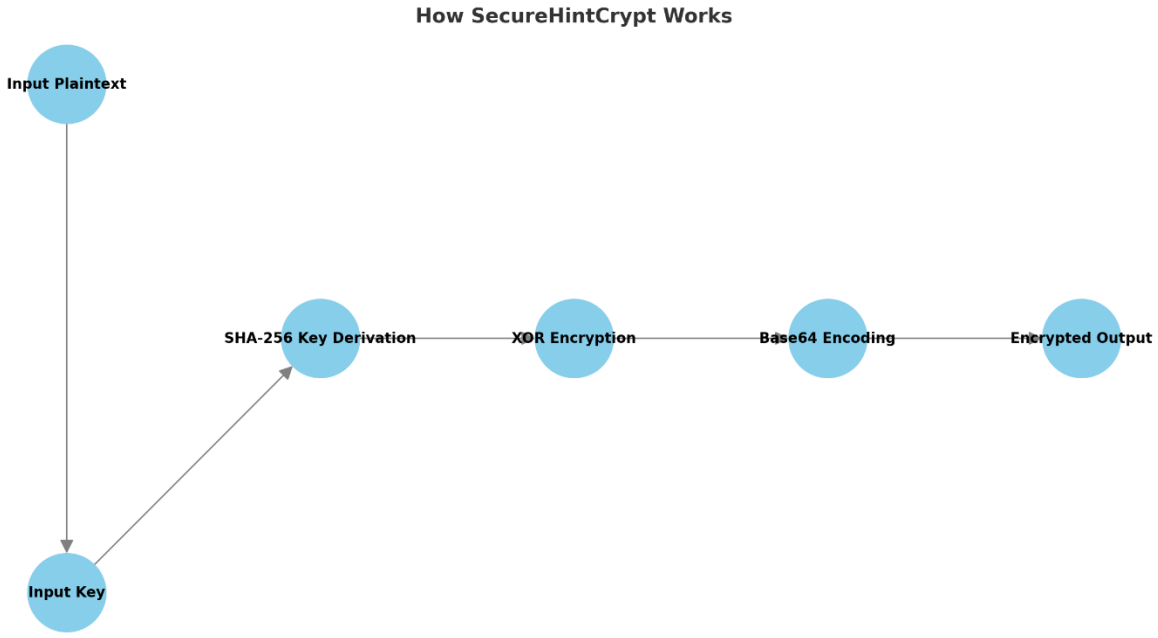
Algorithm Paper

Algorithm Details

Encryption

The encryption process involves the following steps:

1. Hash the provided key using SHA-256 to generate a fixed-length key.
2. XOR each character of the plaintext with the corresponding byte of the hashed key, cycling through the key if necessary.
3. Encode the resulting byte array into a Base64 string.

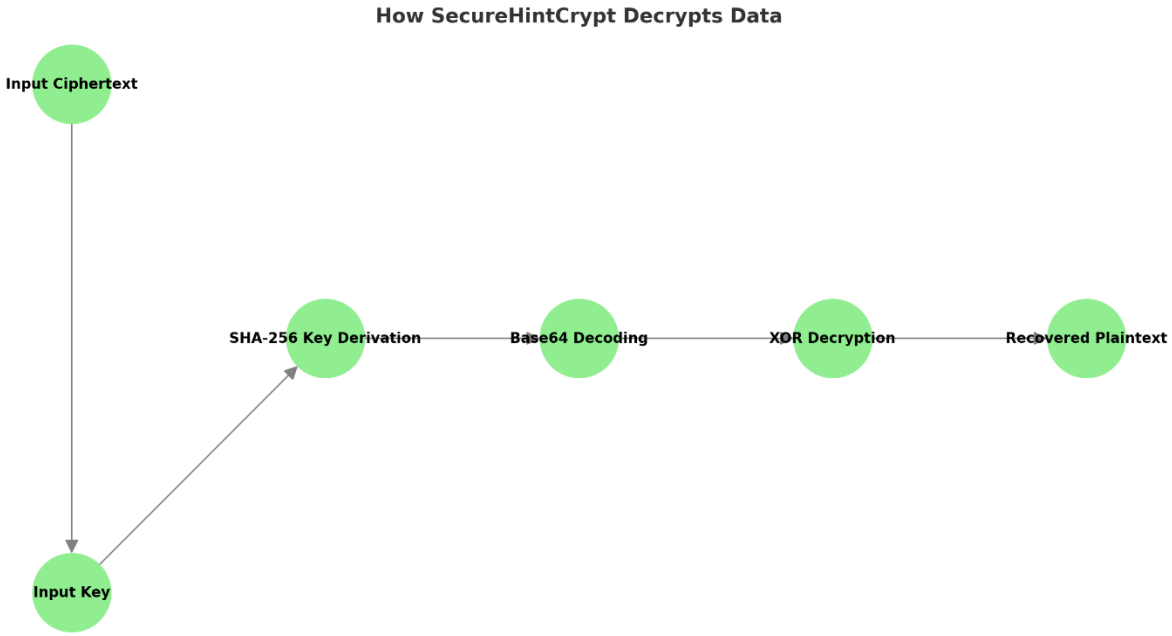


Algorithm Paper

Decryption

The decryption process reverses the encryption steps:

1. Hash the provided key using SHA-256 to generate the same fixed-length key.
2. Decode the Base64 string into its original byte array.
3. XOR each byte of the ciphertext with the corresponding byte of the hashed key, cycling through the key if necessary, to retrieve the plaintext.



Algorithm Paper

Implementation

Below is the Python implementation of the SecureHintCrypt algorithm:

```python
import hashlib
import base64

def custom_encrypt(plaintext, key):
    key = hashlib.sha256(key.encode()).digest()
    ciphertext = bytes([ord(c) ^ key[i % len(key)] for i, c in enumerate(plaintext)])
    return base64.b64encode(ciphertext).decode('utf-8')

if __name__ == "__main__":

    hint = "No Hint for you :)"
    key = "No Key for you :)"

    encrypted_hint = custom_encrypt(hint, key)
    print(f"Encrypted Hint: {encrypted_hint}")
```

Security Analysis

Strengths

- **Key Derivation**: The use of SHA-256 ensures that the key is uniformly distributed and resistant to brute-force attacks.
- **XOR Obfuscation**: Provides a simple yet effective means of encrypting data.
- **Base64 Encoding**: Ensures compatibility with text-based storage and transmission systems.

Limitations

- **Key Management**: The security of the algorithm relies heavily on the secrecy of the key.
- **Known Plaintext Attacks**: If an attacker knows part of the plaintext, they could deduce information about the key.

Algorithm Paper

Use Cases

- **Cybersecurity Challenges**: Securely encode hints or messages for participants.
- **Lightweight Encryption**: For scenarios where simplicity and efficiency are prioritized over advanced cryptographic features.

Conclusion

SecureHintCrypt provides a robust, efficient, and easy-to-implement solution for secure hint transmission in cybersecurity challenges. By combining hashing, XOR, and Base64, it ensures that sensitive information remains protected while being accessible to authorized users.

Future Work

Future iterations of SecureHintCrypt could explore:

- Incorporating stronger encryption methods for enhanced security.
- Developing tools for automated key management.
- Implementing additional layers of obfuscation to counteract potential attacks.

License

Acknowledgments

Special thanks to the cybersecurity community for inspiring me to development this algorithm.