CSE350-
NETWORK SECURITY
ASSIGNMENT NO. 1

Project 1

By –Angadjeet (2022071)
Arav amawate(2022091)

Introduction

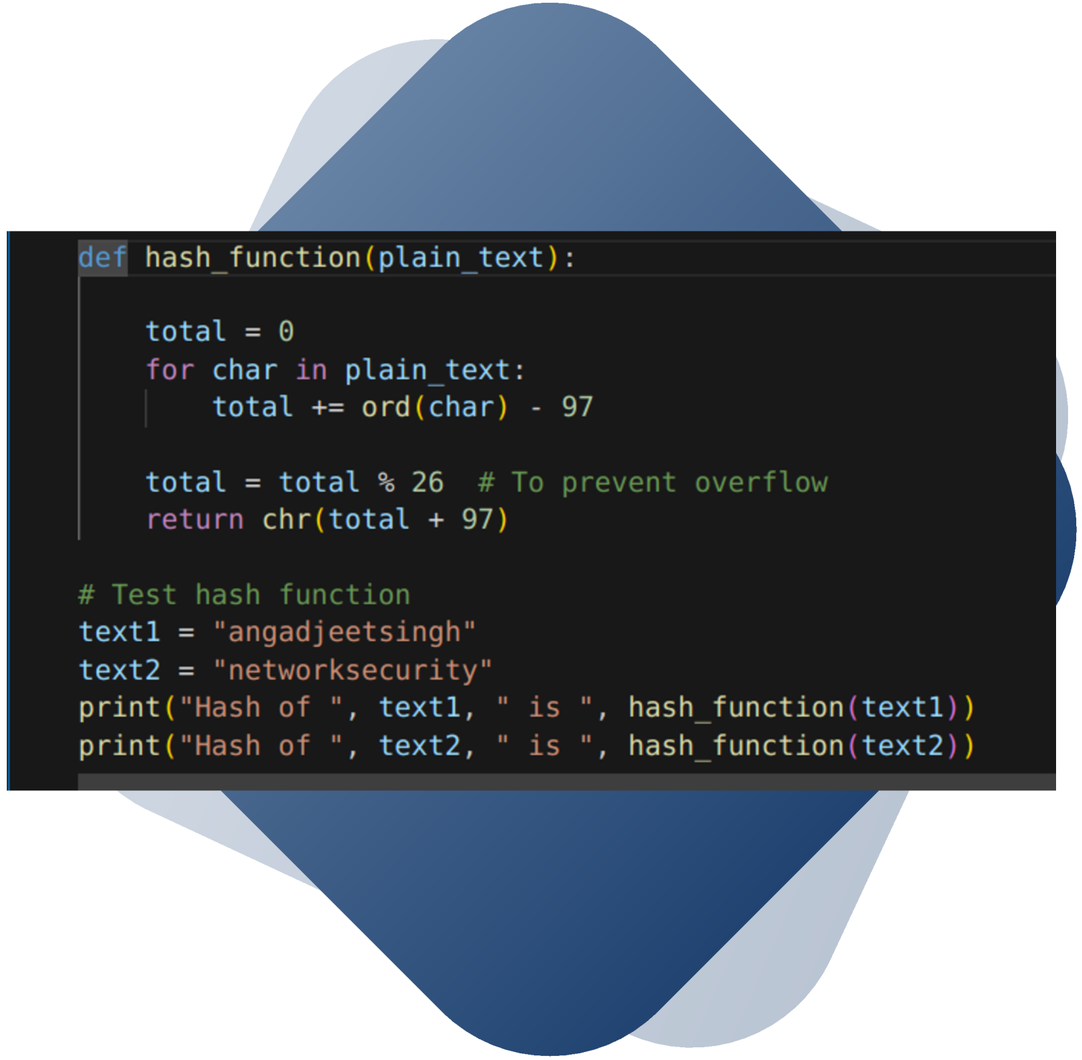
- The Vigenère cipher is a polyalphabetic substitution cipher.
- Enhances security by using a repeating key.
- Objective: Implement encryption, decryption, and perform a brute-force attack to recover the key.

System Overview

- **Encryption:** Uses a repeating key to transform plaintext.
- **Decryption:** Recovers the plaintext using the same key.
- **Hash Validation:** Ensures decrypted text is recognizable.
- **Brute-Force Attack:** Iterates over all possible 4-letter keys to find the correct one.

Hash Function & Validation

- **Purpose: Ensures decrypted plaintexts are valid and recognizable.**
- **Implementation: Computes a simple sum modulo 26:**



```python
def hash_function(plain_text):

    total = 0
    for char in plain_text:
        total += ord(char) - 97

    total = total % 26  # To prevent overflow
    return chr(total + 97)

# Test hash function
text1 = "angadjeetsingh"
text2 = "networksecurity"
print("Hash of ", text1, " is ", hash_function(text1))
print("Hash of ", text2, " is ", hash_function(text2))
```

```python
# The encryption algorithm for the Vigenère cipher is as follows:
def encryption(plain_text, key):
    cipher_text = []

    # CErtain assertions to ensure that the input is in the correct format
    assert type(plain_text) == str, "plain_text should be a string"
    assert plain_text.isalpha() and plain_text.islower(), "plain_text should only contain lowercase English alphabets"

    # Verify if the key is a string and only consists of lowercase English alphabets
    assert type(key) == str, "key should be a string"
    assert key.isalpha() and key.islower(), "key should only contain lowercase English alphabets"

    for i in range(len(plain_text)):
        # Convert letters to numbers (a=0, b=1, ..., z=25)
        m = len(key)
        plain_text_ascii_code = (ord(plain_text[i]) - 97)
        key_ascii_code = (ord(key[i % m]) - 97)

        # Vigenère cipher formula
        cipher_text_ascii_code = (plain_text_ascii_code + key_ascii_code) % MOD_P

        # Convert numbers to letters
        cipher_text.append(chr(cipher_text_ascii_code + 97))

    return "".join(cipher_text)
# Test encryption function
text = "angadandaravarehackers"
print("Encryption of " +  text + " with key " + global_key + ":", encryption(text,global_key))
```

Encryption Process

- Formula: C_i = (P_i + K_{i mod m}) mod 26
- Steps:
    a. Convert plaintext to numeric values.
    b. Use key characters cyclically for substitution.
    c. Convert numeric values back to characters.

```python
    # Vigenère cipher decryption
    def decryption(cipher_text, key):
        plain_text = []

        # Verify input
        assert type(cipher_text) == str and cipher_text.isalpha() and cipher_text.islower(), "Invalid ciphertext"
        assert type(key) == str and key.isalpha() and key.islower(), "Invalid key"

        for i in range(len(cipher_text)):
            m = len(key)
            c = (ord(cipher_text[i]) - 97)
            k = (ord(key[i % m]) - 97)

            # Decryption formula
            p = (c - k) % MOD_P

            # Convert back
            plain_text.append(chr(p + 97))

        return "".join(plain_text)
    text = "nfixqspanjcsnjgenumbek"
    print("Decryption of 'nfixqspanjcsnjgenumbek' with key 'nscx':", decryption(text,global_key))
```

Decryption Process

- Formula: P_i = (C_i - K_{i mod m}) mod 26
- Steps:
    a. Convert ciphertext characters to numeric values.
    b. Use key to reverse the transformation.
    c. Convert back to readable text.

Brute-Force Attack

- Key length is known (4 letters).
- Iterates through 26^4 = 456,976 possible keys.
- Process:
- Generate all 4-letter key combinations.
- Decrypt using each key.
- Validate the result using the hash function.
- Stop when the correct key is found.

```python
def permute(current_key):
    return [''.join(p) for p in permutations(current_key)]

# Validate a key by checking if all plaintexts are recognizable
def is_valid(key, cipher_texts):
    for cipher_text in cipher_texts:
        decrypted = decryption(cipher_text, key)
        if not is_recognizable(decrypted):
            return False
    return True

# Brute-force attack function to find the correct key
total_iterations = 26 ** 4

def brute_force_attack(cipher_texts):
    key_found = False
    attempts = 0  # Track number of attempts

    for i in range(26):
        for j in range(26):
            for k in range(26):
                for l in range(26):
                    attempts += 1  # Increment attempt counter
                    print("Attempting to break the key, try number:", attempts, "out of", total_iterations)
                    print("Current key:", chr(i + 97) + chr(j + 97) + chr(k + 97) + chr(l + 97))

                    # Construct the key using ASCII values
                    key = chr(i + 97) + chr(j + 97) + chr(k + 97) + chr(l + 97)

                    # Try decrypting all cipher texts with the generated key
                    if is_valid(key, cipher_texts):
                        key_found = True
                        print("Key found:", key)
                        return key, attempts

    if not key_found:
        return None, attempts
```

Sample Inputs & Outputs

```
Generated cipher texts:
Cipher text: nfixqbgbgkkktzi
Decrypted text: angadjeetsinghg
Expected Hash: g | Computed Hash: g
Plain text: angadjeetsingh
```

Encrypted text

```
Decrypted texts:
Decrypted text: angadjeetsinghg
Expected Hash: g | Calculated Hash Value: g
Plain text: angadjeetsingh
```

DEcrypted text

```
Ciphertext: awvtbjmpruwovlaa
Decrypted Text: networksecurity
Expected Hash: d | Calculated Hash Value: d
Decryption verified. The text is correct.

End of brute-force attack.
```

Finding the key from brute-force attack

THANKS!