Network Security

Programming Exercise-1

Arnav Singh (2021019) Dharani Kumar S. (2021039)

Algorithm to Pick the Project

$$k=(A1+A2)\%3$$

$$k=(1019+1039)\%3=0$$

Assigned Project:- Monoalphabetic Cipher

I have split the project into four separate independent programmes:-

- Hashing (hash.py)
- Encryption (encrypt.py)
- Decryption (decrypt.py)
- Bruteforce Attack (bruteforce.py)

Hashing

The FNV-1a hash differs from the FNV-1 hash only by the order in which the multiply and XOR is performed:[9][11]

```
algorithm fnv-1a is
    hash := FNV_offset_basis

    for each byte_of_data to be hashed do
        hash := hash XOR byte_of_data
        hash := hash × FNV_prime

    return hash
```

The above pseudocode has the same assumptions that were noted for the FNV-1 pseudocode. The change in order leads t

The output for the standard FNV-1A algorithm is upto 32 bits long, which I have further padded at the left with “0”s to ensure a consistent 33-bit ouput for any and all inputs. This is in line with the restrictions for binary plaintext (All Uppercase ASCII characters are mapped to 3-bit binary strings)

Encryption

```
for i in key:
    if i not in universal_set or len(key)!=len(universal_set) or set(key)!=set(universal_set):
        print("The Entered Key is not satisfactory")
        exit()
print("Key is acceptable\n")
plaintext+=bintotext(hash_fnv1a(plaintext))
hmap={}
for i in range(len(universal_set)):
    hmap[universal_set[i]]=key[i]
print(hmap)
plaintext="".join([hmap[i] for i in [j for j in plaintext]])
```

The user is asked to enter an alphabetical key which is mapped to the ASCII Characters A-H in the same order. The key is then applied on the plaintext (original_text+cipher), and the result is printed on the terminal.

Decryption

```
    exit()
dmap={}
for i in range(len(universal_set)):
    dmap[received_key[i]]=universal_set[i]
received_text="".join([dmap[i] for i in [j for j in received_text]])
pure_text=received_text[0:len(received_text)-11]
hash_text=received_text[len(received_text)-11:]
if hash_fnv1a(pure_text)==texttobin(hash_text):
    print("Succcess")
    print(pure_text)
    print(texttobin(pure_text))
```

The user is asked to enter the received text and key. Next, the key is applied on the cipher-text, and upon successful verification of the cipher text, the original text is printed on the terminal.

Bruteforce Attack

```
possible_keys=permutations(universal_set, 8)
possible_keys=[''.join(p) for p in possible_keys]
for possible_key in possible_keys:
    fmap={}
    for i in range(len(universal_set)):
        fmap[possible_key[i]]=universal_set[i]
    temp_text="".join([fmap[i] for i in [j for j in intercepted_text]])
    pure_text=temp_text[0:len(temp_text)-11]
    hash_text=temp_text[len(temp_text)-11:]
    if hash_fnv1a(pure_text)==texttobin(hash_text):
        print("Succcess")
```

The user is asked to enter the intercepted text. The program tries 8! different keys to verify the “decrypted” cipher’s hash, which upon successful verification, leads to the original message being printed on the terminal.

Thanks!