# Lab Report

Name: Guo Yuchen

Student ID: 1004885

## Justify the correctness of your implementation of the RC4 algorithm

Implementation:

*def* KSA(*key*):

    S = list(range(256))

    # Add KSA implementation Here

    for i in range(256):

        S[i] = i

    j = 0

    for i in range(256):

        j = (j+S[i] + *key*[i % len(*key*)]) % 256

        S[i], S[j] = S[j], S[i]

    return S

*def* PRGA(*S*):

    # Add PRGA implementation here

    K = 0

    i = 0

    j = 0

    while True:

        i = (i+1) % 256

        j = (j+*S*[i]) % 256

*S*[i], *S*[j] = *S*[j], *S*[i]

        t = (*S*[i]+*S*[j]) % 256

        K = *S*[t]

        yield K

test: run the attached file “testRC4.py”, the result OK proved that the implementation is correct.

## The cracked payload and ICV of one broadcast packet

Choose the packet No.45 from the cap file:

# From Number 45:

    CIPHERTEXT = "4d4979775670b2417d2c0a9a37dee9e1a8dc1b0a55c0a4d554a686a48a231e21f55e44c6da9f34b52e96f6244fb2416a5021dd91db76"

    IV = "d6043f"

    ENCRYPTED\_ICV = "de3e7fca"

    KEY = "1F1F1F1F1F"

    ciphertext =binascii.unhexlify (CIPHERTEXT+ENCRYPTED\_ICV)

    # Use RC4 to generate keystream

    key =binascii.unhexlify(IV+KEY)

    keystream = RC4(key)

    # Cracking the ciphertext

    plaintext = ""

    for i in ciphertext:

        plaintext += ('{*:02X*}'.format(i ^ next(keystream)))

    # Check ICV

    crcle = binascii.crc32(bytes.fromhex(plaintext[:-8])) & *0x*ffffffff

    crc = struct.pack('<L',crcle)

    icv = plaintext[-8:]

    print("Payload: ",plaintext[:-8])

    print("Decrypted CRC: ", icv)

    print("Calculated CRC: ",crc.hex())

The output is:

Payload:  AAAA0300000008060001080006040001000EA66BFB69AC100001000000000000AC1000F0000000000000000000000000000000000000

Decrypted CRC:  6B8FE49D

Calculated CRC:  6b8fe49d

Thus the decrypted crc equals the calculated crc.