**Solution**

1) For the RC4 stream cipher, we use the keys “Key” and “key” on the plaintext “Plaintext”, and the resulting ciphertexts are as follows (Other implementations might give different implementations based on different way they handle leading zeroes and the mapping of characters to integer ascii values):

*Ciphertext is: bbf316e8d940af0ad3*

*Plaintext is: Plaintext*

*Ciphertext is: 5b0055844afb1e323c*

*Plaintext is: Plaintext*

The number of positions in which they differ is **17**, an incredible amount considering they are only **18** nibbles long (in *hexadecimal notation)*, making the total difference **94.4%.** Going to binary notation, the difference is **40** bits out of **72** bits (**55.56%**), close to the statistical value of 50%, proving the security of RC4 ciphers. Also, since it is a stream cipher, we don’t have the problem of creating a huge key for a huge text, but can still emulate OTP like behaviour. It is also more malleable than common block ciphers.

However, there are still issues. Unlike a modern stream cipher, RC4 does not take a separate nonce alongside the key. This means that if a single long-term key is to be used to securely encrypt multiple streams, the protocol must specify how to combine the nonce and the long-term key to generate the stream key for RC4. One approach to addressing this is to generate a "fresh" RC4 key by hashing a long-term key with a nonce. However, many applications that use RC4 simply concatenate key and nonce; RC4's weak key schedule then gives rise to related key attacks like the *Fluhrer, Martin and Shamir* attack.

2) DES. For plaintext “Aditya”, and the keys 0123456789ABCDEF and 1023456789ABCDEF (1 bit flipped), we get (Note: pass the value **verbose =** **true** in the program to see outputs of all rounds):

*Encrypted text: (Key=0123456789ABCDEF) 5e1e4faf782a0c5d667acd86afe0812299f2793b9a6ea0080884ddd9495e55c3de5b77e38e9a9bd4adf6cb79869ddbdb*

*Decrypted text: Aditya*

*Encrypted text:(Key=1023456789ABCDEF)*

*89d2133821d0871d77dda644a921ff81ea7b7cfa502d25eccee0ffb2898801ea0bf00d55af6f99316cdfd38ceee429da*

*Decrypted text: Aditya*

Difference in *hexadecimal nibbles* is **90/96**, or **93.75%** and in *binary notation*, **180/384** bits or **46.875%** bits are different, again close to the statistical value of 50%.

**Strength:** Strong confusion and diffusion properties, primarily due to the S-boxes (Substitution boxes). Also, encryption and decryption use the same keys and the same function, making it hardware friendly and cost effective. It also has the completeness property, that is each bit of ciphertext depends upon multiple bits of plaintext.

**Weakness**: The primary weakness is the 56-bit key size which is vulnerable to brute force attacks-In 1999, a DES key was publicly broken in less than 23 hours, and it would take even lesser time with today’s technology (a practical mitigation is to move to triple DES). Differential cryptoanalysis, linear cryptoanalysis and Davies’ Attack also exist which is supposed to break DES more effectively than brute force, but they are generally considered infeasible in practice. Also, S-box be made to produce same output with two chosen input.