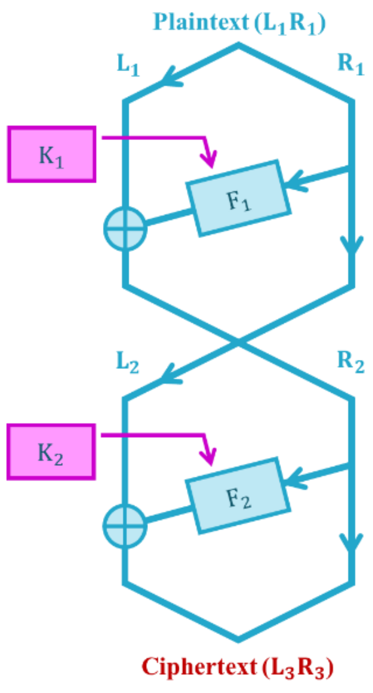
6. Ciphers and Stack Vulnerabilities

COMP6441 • KC Notes

6.1 Ciphers

- Claude Shannon – a good cipher has two things:
    - **Confusion**: the relationship between the **key and ciphertext** is mysterious
    - **Diffusion**: the relationship between the **plaintext and the ciphertext** is mysterious
        - If someone changes one bit of the plaintext, we want at least half of the ciphertext to change
        - Most ciphers **avalanche**, so that it's hard to go backwards
- Dooke – looked at random numbers so that text could be properly encrypted
    - Something may **look chaotic, but plotted in 3D** can form regular lattice
    - Non-randomness helps make random numbers
        - Ciphers make **systematic changes** that are hard to undo
- **Lucifer**: a block cipher used by banks to transmit data
    - Was submitted as the Data Encryption Standard (DES)
    - Conjured up in a **competition by NIST** to submit good encryptions, run every five years
    - NSA changed numbers in the system (there's discipline and knowledge in Lucifer?) – they had made it resistant to differential cryptanalysis.
- **Fiestal**: **symmetric** structure where the plaintext is split into two – the first joined with a key and put into a function F, and then XORed with the other. This is **repeated** with another key.
    - Each round changes **half the bits** (the other half gets changed in the next round)



Encrypting

1. Split plaintext into two

$L_2 = R_1$

Join R_1 and K_1 and put into F_1
XOR function with L_1
$R_2 = L_1 \oplus F_1(R_1, K_1)$

$L_3 = R_2$
$R_3 = L_2 \oplus F_2(R_2, K_2)$

Decrypting

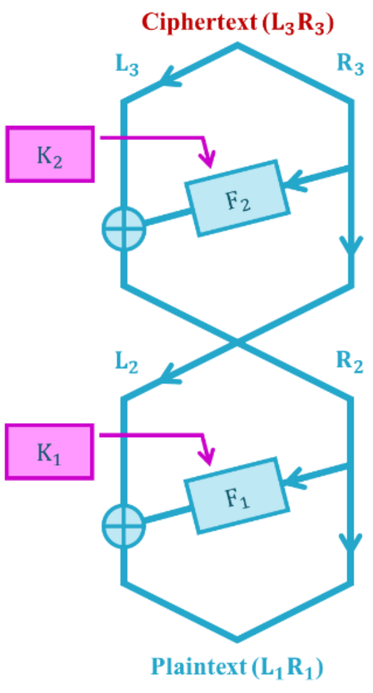
1. Split plaintext into two

$L_2 = R_3$

Join R_1 and K_1 and put into F_1
XOR function with L_1
$R_2 = L_3 \oplus F_3(R_3, K_3)$

$L_1 = R_2$
$R_1 = L_2 \oplus F_2(R_2, K_2)$

Note that encrypting and decrypting are nearly the same!
Also, because XORing twice results in the same string, your function can be anything (doesn't need to be invertible).



    - Because XOR is invertible, the function does not need to be invertible.
    - The same key is used to encrypt and decrypt
    - Relies on symmetric cryptography
    - **AES**: Advanced Encryption Standard works in a similar way with multiple functions.
        - AES-128 has 10 rounds, 192 with 12, 256 with 14
        - **SP network**: AES iterates between substitution and permutation ciphers.

6.2 Assymetric and Symmetric

- **Assymetric/Public key cryptography** is slow
    - It relies on **complexity in maths**
    - The public key may leak information about the private key
    - BUT it solves the key distribution problem
    - One way mathematical problems need to be **proven to be hard** to go backwards
- **Symmetric key cryptography** is fast
    - It relies on **complexity in randomness**
    - **Safes** are an example of a symmetric key: they are cheap and easy to encrypt, but they are certified to a time limit, e.g. 2 hours, 3 hours, rather than keeping it out
- **Side channel attack**: looking at metadata to attack a cipher/system
    - DES: can be attacked by monitoring power supply, latency, memory changes

6.3 Red Teaming (Guest Speaker: Finbar)

- Red team attacks, blue team defends
    - Red team is the enemy, company is being attacked, and the company is tested in how they respond to scenarios
    - Red team trains the blue team
- Focuses on the most important assets, and shares where it was easy or difficult to get through

6.4 Stack Vulnerabilities (Guest Speaker: Finbar)

- **Morris worm** (1988): one of the first worms distributed in the Internet, a buffer overflow
    - Independently discovered buffer overflow exploitation by Thomas Lopatic in 1995
    - 1996: PHRACK article, Smashing the Stack for Fun and Profit
- **Stack canaries**: a **tamper seal** before a return pointer – to overwrite the return pointer, you also have to overwrite the canary value, which is random
- **ASLR**: Address Space Layout Randomization, data within the memory space are shifted around
    - **PIE**: Position-independent executables, similar to ASLR
- **NX**: No-eXecute, to prevent code from being executed in the stack
- **Shadow stacks**: Hardware based isolated return pointer – it checks if the return address is the same as it was when function is called, before going back.