Project 1 Report

# Design Explanation

* + Meta-data design
    - Meta-data is stored in the 0th file. 0 to 127 bytes are username, 128 byte to 256 byte are the encrypted data for length of file and hashed value of random initial vector with password. The next 32 bytes are HMAC value to check integrity.
  + User authentication
    - To modify the file, the password hash value and the stored password hash value must match.
  + Encryption design
    - Files are encrypted block by block. As writing a portion of block requires fetching the entire block, the content of the entire block under the starting position is decrypted, modified, then encrypted and stored back. For every 16 bytes (128bit) in each block, the initial vector is incremented and reset when encrypting the next block.
  + Length hiding
    - All files when writing are padded to 1024 bytes, so one cannot overserve the difference between a half full block and a complete block.
  + Message authentication
    - Every time when a file is modified, the length and HMAC values are recalculated to ensure its modification is authorised.
  + Efficiency
    - As it is focus on the storage efficiency, it first finds the next available room in the requested location then make the modifications to the file.

# Pseudo Code

* 1. create(String file nameString user name, String password)
  + padUsername()
  + pad and hash password()
  + create secretdata{length of file, password hash}
  + saves{padded username, secret data, HMAC result of the first 2 items}
* 2. String findUser(String file name)
  + Reads the 0 block for metadata
  + Extract username from 0-th slot
* 3. int length(String file name, String password)
  + Reads the 0 block for metadata
  + Sets initial vector
  + Decrypt secret data containing the length information
  + Check whether the password entered has a same hash value as the one stored
  + If password correct, return length
* 4. byte[] read(String file name, int starting position, int length, String password)
  + Read the entire block from the designated starting position
  + Decrypt the content, store the information for output
  + Output the decrypted content
* 5. void write(String file name, int starting position, byte[] content, String password)
  + Use length function to check passcode correctness
  + Locate the block where the starting position is in
  + Calculate the ending block using the length information
  + Even if partial of block is rewritten, the entire block is rewritten
    - Store the unmodified contents
    - Apply modification
  + Encrypt the finished block
  + Change the length portion and HMAC of metadata
    - Decrypt secretData field of metadata
    - Modify length
    - Recalculate HMAC
    - Store metadata
* 6. void cut(String file name, int length, String password)
  + Use length function to check passcode correctness
* 7. boolean check integrity(String file name, String password)
  + Compare HMAC stored calculated with the stored value

# Design Variation

* 1. Suppose that the only write operation that could occur is to append at the end of the file. How would you change your design to achieve the best efficiency (storage and speed) without affecting security?
  + To achieve the best efficiency, one can try to append new content to the last half of the last block, if the second half of the last block is available. Since finding the second half is easy with fixed block size, and no more than 512 byte of storage saves time in searching for the next available slot.
* 2. Suppose that we are concerned only with adversaries that steal the disks. That is, the adversary can read only one version of the same file. How would you change your design to achieve the best efficiency?
  + Frequently change the order which partial files are arranged.
* 3. Can you use the CBC mode? If yes, how would your design change, and analyze the efficiency of the resulting design. If no, why?
  + It is possible, but random access will be unavailable since previous blocks needs to be decrypted before reading and writing to the desired block.
* 4. Can you use the ECB mode? If yes, how would your design change, and analyze the efficiency of the resulting design. If no, why?
  + It is possible. If using ECB mode, we just need to use the same set of key to encrypt instead of a random number generator produced initial vector.

# Paper reading

From “Why Cryptosystems Fail,” it shows that most cryptosystems are vulnerable because the designers of the system used a wrong threat model during design. Designers usually assumes that their system will be implemented by some expertise who is skilful in building security system or with the help from some consultant. However, due to the lack of motivation on the company’s side, unskilled personnel may be selected to implement the system. As a result, frauds are mostly committed by implementation errors and management failures instead of cryptanalysis or technical attacks. Since crypto community seldom researches and learns from the mistakes found in the system, it blocks any feedback usable by the cryptosystem designers when reviewing the threat model for a new design, leading to another failed cryptosystem design. Although there are standards helping companies to choose the security infrastructure their company need, those standards were not considering human errors either, allowing human mistakes to break these cryptosystems easily even if the system has been certified some ultimate level security. Because of irrelevance between the target research force of computer security and the real needs, a paradigm shift is required so the new goals in computer security encompass software engineering ideas, so human error during the operation of the system can be considered in the threat model and incremental improvement can be made in the progression of cryptosystem.

From “Intercepting Mobile Communications: The Insecurity of 802.11,” the article shows that WEP protocol fails to deliver its security goals in confidentiality, access control, and data integrity, and that the security the protocol relies on can be defeated either due to short key length (40-bit for the initial version), or exploitable WEP design model. WEP attacks are feasible to mount because it does not safeguard against resourceful attackers who may have the computational power to brute-force WEP, and the hardware equipment to monitor and inject Wi-Fi traffic is readily available in market. With the widely available equipment, the attacker can tap into any 2.4GHz wireless communication and search for keystream reuses, especially some low-valued initial vector keys as some devices zeroed the initial vector offset whenever it is unplugged. With 2 encrypted packets with same IV, they are XORed to reveal the XORed value of the plaintext of the 2 packets. Since some fields of IP traffic is predictable, once plaintext of either packet is decrypted, then both packets’ plaintext can be decrypted. With multiple ciphertext matching their corresponding secret key, the attacker can build a full decryption dictionary, completely busting the WEP protocol open. Since the plaintext is consist of original message and CRC checksum, and CRC checksum is not cryptographically secure, attacker can modify the sender’s message, or even inject his own message in the sender’s behalf while tapping the traffic between the sender and the receiver through IP redirection attack or reaction attack. To counteract, network administrator can place the wireless network outside the firewall and employ VPN to access the internal network, or improve the key management such that every host has its own dynamic encryption key that frequently changes. The failure of WEP shows that one should reuse past design and make incremental improvement base on the previous design when crafting a new design, then let the public to review such a design so that the design is fully tested against flaws.