Applied Cryptography

1 Substitution Cyphers

Replaces each letter of the alphabet with another letter, e.g. ROT13 is a popular basic example. ROT cyphers are easy to break as you just iterate over all keys and fuzzy string search a word list. There are lots of variants:

- Monoalphabetic - Fixed substitution
- Polyalphabetic - Change substitution rules in different parts of the message
- Polygraphic - Substitute with groups of letters

2 Encryption and Decryption

- Both keys the same - Symmetric key
- Both keys are different - Asymmetric key or public key cryptography
- Range of possible values for key - keyspace
- Range of possible values for cipher - cipherspace

3 Block Ciphers

- Symmetric key encryption method is typically used for files
- Encrypts blocks of text at a time, rather than bits of text (stream ciphers)
- DES encrypts 64-bit blocks at a time
- AES encrypts 128-bit (or bigger) blocks at a time
- Developed to eliminate the chance of encrypting identical data the same way: the ciphertext from the previous block is fed into the algorithm for computing the next block
- Also uses an initialisation vector such that the same message encrypted multiple times will be different
- Can't be broken easily with GPUs

4 ECB vs Non-ECB modes

ECB - Electronic code book, doesn't use initialisation vector
ECB mode means that the sensitive data can often still be decrypted, especially when it comes to images as some errors can be compensated with by the human brain.

Initialisation vectors are much better, ECB mode is not secure

5 Hacking AES-256

AES-256 is currently regarded as one of the most secure block cipher algorithms. To brute force you would need 2^{256} guesses, which would take longer than the age of the universe

There are side channel attacks on this, take a secure implementation and look at timings and outputs etc which gives another means to see how the implementation is going. As the algorithm will do different things if a password is correct or not, it is easy to look at the timing to find the answer.

6 Storing Passwords

Storing Passwords in plain text is not good

- If someone obtains database of user IDs/passwords then all users are exposed
- We should design it that, even if there is a flaw in the system security, the password should be hard to find
- We can hash the passwords and check the hashes match instead
- Will storing our passwords as a list of hashes, which can't be inverted, make us secure?
- Most passwords are
  - Not random characters
  - Not arbitrary length
  - Have some structure to them
- For example 9 characters would take 2 years to brute force

7 Hash Functions

- Any function that can map data of arbitrary size to a fixed size
- Different applications require hash functions with different properties
- Cryptographic hash functions should guarantee these properties
  - Deterministic (the same message always should result in the same hash)
  - One-way function (can't reverse output to input)
  - No collisions (no two input messages produce the same output)
  - Avalanche effect (changing 1 bit in the input should produce a massive change in the output)
  - Fast? (depends on the application, password storage should be slow to make brute forcing harder)

8 Precomputed hash tables

By precomputing and crowd-funding the hashes, the lookups would be much faster. Without compression this would be too big
Rainbow tables are a combination of an algorithm and a data structure.

- Have a mapping between hashes and plaintext
- Hash the plaintext, apply reduction function (truncate e.g)
- Keep going 100,000 times
- The input and output hash contains all the information of 100,000 times

A rainbow table maps hashes to their plaintext equivalent
To query the rainbow table

1. Iterate 100,000 times. Look for the hash in the sorted list of final hashes
2. If not found reduce the hash into another plaintext, and hash the new plaintext
3. If it is found, the chain for which the hash matches the final hash contains the original hash. You can now go from the start of the chain to recover the secret plain text

9 Salts

- Using rainbow tables we can recover passwords within minutes and retain reasonable storage requirements
- So we introduce a ”salt” which is a random string stored as plain text alongside the hash, but we can compute the hash by

$$\text{hash}=H(\text{salt}+\text{password})$$

- Two users with the same password will not have different hashes, as they will have different randomly generated salts
- For 32-bit salts, you would not need to pre-compute and query 2^{32} rainbow table databases (for each salt value) making such hacking approaches infeasible

10 Coursework

Over 25 vulnerabilities
Try and solve the mystery in the system