Week 5: Attacks & Security

BRUTE FORCE ATTACK

Brute Force Attack is the simplest form of cryptanalysis, where the attacker tries *all possible keys* until the correct one is found.

Assumptions:

- The encryption algorithm is known.
- The keyspace is small enough to try all keys.
- The attacker can recognize the correct plaintext (e.g., English words).

FREQUENCY ANALYSIS

Frequency analysis is a cryptanalysis technique based on how often letters appear in a given language (typically English).

In English:

- **E** is the most common letter (~12.7%)
- Followed by **T**, **A**, **O**, **I**, **N**, **S**, **H**, **R**...

This method is especially useful against:

- **Monoalphabetic substitution ciphers**, where each letter in the plaintext is replaced by exactly one letter in the ciphertext.

If the ciphertext is long enough, the frequency of letters in the ciphertext will reflect the frequency in the plaintext. By matching the frequencies, we can guess the original letters.

Steps in Frequency Analysis Attack

1. **Count frequency** of each letter in the ciphertext.
2. **Rank** them in order of appearance.
3. **Map** the most frequent cipher letters to most frequent English letters.
4. **Try substitutions** and iterate — some guesswork and context knowledge helps.
5. Refine based on common words, patterns, and digraphs (like **TH**, **HE**, **IN**, **ER**).

CHOSEN-PLAINTEXT ATTACK (CPA)

In a **Chosen-Plaintext Attack**, the attacker can **ask for the encryption of plaintexts they choose**, and use the resulting ciphertexts to gain information about the key or the encryption process.

This is different from just passively observing plaintext–ciphertext pairs (known-plaintext attack) — here, the attacker actively *chooses* the input!

Example: CPA on Vigenère Cipher

Suppose you input:

- `AAAAA` → Output: `K E Y K E`
- You now know that:
  `A + K = K` → key[0] = K
  `A + E = E` → key[1] = E
  `A + Y = Y` → key[2] = Y

Just by submitting `AAAAA`, you recovered the key!
If the key is repeated every 3 characters, you can find its length too.

In Modern Cryptography

CPA is taken **very seriously** — secure systems like AES or RSA must be CPA-resistant.

That's why we define **semantic security**:

- A cipher is **semantically secure** if an attacker **cannot learn anything** about plaintext, even when allowed to encrypt plaintexts of their choice.

Defense Against CPA

- Use **randomized encryption**: like adding IVs (initialization vectors)
- **Padding schemes** like OAEP (in RSA)
- Use **CPA-secure modes**: CBC with random IV, GCM, etc.

TIMING ATTACKS

A **timing attack** is a type of **side-channel attack** that doesn't break the algorithm itself — instead, it exploits information from **how long** a system takes to perform cryptographic operations.Even tiny differences in processing time (measured in nanoseconds or milliseconds) can **leak information about secret keys** or internal states.

How It Works

Let’s say you have a function that compares passwords like this:

```
def check_password(input, real_password):
    for i in range(len(real_password)):
        if input[i] != real_password[i]:
            return False
    return True
```

This checks the password character by character.
If a user enters `'K'`, and the correct password is `'KEY'`, it takes more time than if they entered `'Z'`, because the first character matched.

An attacker could measure time to find out how many characters matched, and use this to reconstruct the password/key **one character at a time**.

Countermeasures

1. **Constant-time algorithms** — make sure all operations take exactly the same time regardless of input or key.
2. **Blinding techniques** — introduce randomness in the computation.
3. **Avoid branching on secret data** — avoid if/else based on key bits.

Simulating Attacks

BRUTE FORCE - CAESAR CIPHER

To **simulate** a brute force attack on a Caesar cipher:

- You will be given a ciphertext.
- You'll try all 25 possible shifts (since Caesar uses mod 26).
- The correct plaintext will be recognized manually or automatically using English word detection

Analysis

- Brute force works well for **Caesar** because the keyspace is tiny (only 25).
- It becomes **ineffective** when the keyspace grows large (e.g., AES-128 → 2^{128} keys).
- In real attacks, attackers combine brute-force with **dictionary attacks** or **statistical heuristics**.

FREQUENCY ANALYSIS - MONOALPHABETIC CIPHER

Given a ciphertext encrypted using a **monoalphabetic substitution cipher**, we'll:

1. Count the frequency of letters in the ciphertext.
2. Match the top letters to the most frequent English letters.
3. Build an approximate decryption.

CPA - VIGINERE CIPHER

In this simulation, we will:

- Encrypt a **chosen plaintext** using the Vigenère cipher.
- Use the resulting ciphertext to recover the key.

This models a real **chosen-plaintext attack**, where the attacker submits custom plaintexts (like `AAAAA...`) to uncover the encryption key.

TIMING ATTACK

A **timing attack** exploits the **time it takes** a system to respond. Insecure programs might take slightly **longer** to respond when certain conditions are met — for example:

- When more characters of a password match.
- When the decryption logic takes longer based on key comparisons.

These **tiny delays** (even in milliseconds or microseconds) can leak information.

Imagine a login function like this:

```
bool insecure_compare(string input, string correct) {
   for (int i = 0; i < input.length(); ++i) {
      if (input[i] != correct[i])  return false;
      delay_microseconds(50);  // Leaks time per matched character! }
   return true;}
```

An attacker can:

- Measure the response time.
- Guess the correct value **character by character**, by observing if the time increases.

Timing attacks:

- Are extremely dangerous in **cryptographic APIs**, login functions, or token verification.
- Can be performed **remotely** via network latency measurements.