ISS Analysis of Attacks

Name: Jasraj Shendge
Reg no: 229301558
Section: CSE-6H

Introduction

Cryptography is responsible for protecting electronic communications, yet most past and even some present-day cryptographic algorithms contain weaknesses susceptible to attack via powerful cryptanalytic methods. This report covers three such attacks:

•Vigenère Cipher Breaking by Kasiski Examination and Frequency Analysis – Vigenère Cipher, which was thought to be secure, is broken by looking for repeated patterns in ciphertext (Kasiski Examination) and using statistical frequency analysis to infer the key.

•Meet-in-the-Middle Attack on Double DES (2DES) – Although Double DES was intended to improve security, the Meet-in-the-Middle attack undermines its effective strength by precomputing intermediate states of encryption and renders it just slightly more secure than single DES.

•Pollard's Rho Algorithm for Integer Factorization – Some public-key cryptosystems like RSA are based on the difficulty of factoring large numbers. Pollard's Rho is a fast probabilistic factorization algorithm that can factor weak RSA keys by discovering non-trivial factors.

All these attacks show how theoretical weaknesses can be reduced to working exploits, further emphasizing the need for strong cryptographic construction. We explain the mechanism of each attack below, including code examples and their implications in real life.

1. Vigenère Cipher Breaking: Kasiski Analysis and Frequency Attack

Overview

The Vigenère Cipher is a polyalphabetic substitution cipher that employs a repeating keyword to encrypt a message. Every letter in the plaintext is shifted according to the corresponding letter in the key, wrapping around the key if necessary. Although more secure than basic substitution ciphers, the Vigenère Cipher is susceptible to several cryptanalysis methods, such as Kasiski Examination and Frequency Analysis on Key Lengths.

Kasiski Examination

ISS Analysis of Attacks

Attack Explanation

Kasiski Test is a technique used to find the length of the key of a Vigenère-encrypted message by finding repeated substrings in the ciphertext. As the key repeats, the same plaintext segments could be encrypted the same way at fixed intervals. By examining the interval between the repetitions, we can infer the likely key length.\

Code Implementation

```
import re

from collections import Counter


def find_repeated_sequences(ciphertext, min_length=3):
    repeated_sequences = {}
    for i in range(len(ciphertext) - min_length + 1):
        seq = ciphertext[i:i + min_length]
        if seq in repeated_sequences:
            repeated_sequences[seq].append(i)
        else:
            repeated_sequences[seq] = [i]


    return {k: v for k, v in repeated_sequences.items() if len(v) > 1}


def find_key_length_distances(repeated_sequences):
    distances = []
    for positions in repeated_sequences.values():
        for i in range(len(positions) - 1):
            distances.append(positions[i + 1] - positions[i])


    return Counter(distances)


ciphertext = "LXFOPVEFRNHR"  # Example ciphertext
repeated_sequences = find_repeated_sequences(ciphertext)
distance_counts = find_key_length_distances(repeated_sequences)
print("Repeated Sequences:", repeated_sequences)
print("Distance Counts:", distance_counts)
```

Impact

By finding the greatest common divisor (GCD) of the distances between repeated sequences, attackers can estimate the key length. Once the key length is known, further analysis can be applied to decrypt the message.

Frequency Analysis on Key Lengths

Attack Explanation

Once the key length is determined, the ciphertext can be split into groups corresponding to each letter in the key. Each subset behaves like a simple substitution cipher, allowing letter frequency analysis to recover the key.

Code Implementation

```
import string


def frequency_analysis(ciphertext, key_length):
    frequency_tables = [{} for _ in range(key_length)]
    for i, char in enumerate(ciphertext):
        if char in string.ascii_uppercase:
            idx = i % key_length
            if char in frequency_tables[idx]:
                frequency_tables[idx][char] += 1
            else:
                frequency_tables[idx][char] = 1


    return frequency_tables


key_length = 3  # Assumed from Kasiski Examination
frequencies = frequency_analysis(ciphertext, key_length)
for i, freq in enumerate(frequencies):
    print(f"Letter Frequencies for Position {i}:", freq)
```

Impact

By comparing letter frequencies with known English letter distributions, the key can be reconstructed letter by letter. Once the key is identified, decrypting the entire ciphertext becomes straightforward.

2.Meet-in-the-Middle Attack on 2DES

ISS Analysis of Attacks

Exploiting Double Encryption

Double DES (2DES) was introduced to strengthen single DES by encrypting plaintext twice with two different 56-bit keys. However, the **Meet-in-the-Middle (MITM) Attack** significantly reduces the effective security of 2DES from **112 bits** to **only 2 × 56 bits** (or **57 bits** of security).

This attack works by dividing the encryption process into two stages:

1. Encrypting the plaintext with the first key (K1).
2. Decrypting the ciphertext with the second key (K2).

Since both processes meet at an intermediate encrypted state, an attacker can precompute all possible first encryptions and match them against all possible decryptions, drastically reducing the complexity.

Code Implementation

```
from itertools import product


def simple_des_encrypt(plaintext, key):
    return (plaintext + key) % 256  # Simplified DES-like operation


def simple_des_decrypt(ciphertext, key):
    return (ciphertext - key) % 256  # Simplified DES-like operation


def meet_in_the_middle_attack(plaintext, ciphertext):
    key_space = range(256)  # Simulating an 8-bit key space for demonstration
    encryption_map = {}

    # First pass: Encrypt plaintext with all possible keys
    for K1 in key_space:
        intermediate = simple_des_encrypt(plaintext, K1)
        encryption_map[intermediate] = K1

    # Second pass: Decrypt ciphertext with all possible keys
    for K2 in key_space:
        intermediate = simple_des_decrypt(ciphertext, K2)
        if intermediate in encryption_map:
            return encryption_map[intermediate], K2  # Found matching keys
```

```
    return None


plaintext = 50  # Example input
ciphertext = simple_des_encrypt(simple_des_encrypt(plaintext, 42), 85)  # 2DES encryption
keys = meet_in_the_middle_attack(plaintext, ciphertext)
print("Recovered Keys:", keys)
```

Impact

By precomputing possible intermediate values and storing them in a table, the MITM attack reduces the complexity of breaking **2DES** from **2^112** to **2 × 2^56**, making it vulnerable to brute-force attacks within practical limits. This is why **Triple DES (3DES)** or more secure encryption methods like **AES** are recommended over 2DES.

3.Pollard's Rho Algorithm for Factorization

Description

Pollard's Rho algorithm is a **probabilistic** integer factorization algorithm that efficiently finds non-trivial factors of a composite number. It is particularly useful in cryptanalysis, as many cryptosystems (such as **RSA**) rely on the difficulty of factorizing large numbers.

The algorithm works by using a pseudo-random function to generate a sequence of numbers modulo n and detecting cycles using the **Floyd's cycle-finding method** (also known as the "tortoise and hare" technique). When a cycle is found, the greatest common divisor (GCD) of the difference between two sequence values and n often reveals a factor.

Attack Explanation

RSA security is based on the assumption that **factoring large semiprime numbers (products of two primes) is computationally hard**. However, Pollard's Rho algorithm can quickly factor smaller numbers and is effective against poorly chosen RSA key sizes.

Steps of Pollard's Rho Algorithm:

1. Choose a random function, typically $f(x) = (x^2 + c) \bmod n$.
2. Use two pointers (slow and fast) to generate a sequence:
    - slow moves one step at a time.
    - fast moves twice as fast.
3. When a cycle is detected (i.e., slow == fast), compute gcd(|slow - fast|, n).
4. If a non-trivial factor is found, return it; otherwise, restart with a different function.

Code Implementation

python

CopyEdit

```
import random
import math

def pollards_rho(n):
    if n % 2 == 0:
        return 2  # Return 2 if the number is even

    # Random function f(x) = (x^2 + c) % n
    def f(x, c, n):
        return (x * x + c) % n

    x = random.randint(2, n - 1)
    y = x
    c = random.randint(1, n - 1)
    d = 1  # GCD

    while d == 1:
        x = f(x, c, n)  # Move one step
        y = f(f(y, c, n), c, n)  # Move two steps
        d = math.gcd(abs(x - y), n)  # Compute GCD

        if d == n:  # Failure, retry with a different c
            return pollards_rho(n)

    return d

# Example: Factoring a composite number
n = 10403  # Example composite number (product of two primes)
factor = pollards_rho(n)
print(f"A non-trivial factor of {n} is {factor}")
```

Impact

Pollard's Rho algorithm is **efficient for small to medium-sized numbers** and is frequently used in practical cryptanalysis. If an RSA modulus N is factored using Pollard's Rho, then the private key can be reconstructed, breaking the encryption.

However, modern RSA implementations use **large prime numbers** (2048+ bits) that make Pollard's Rho impractical. To counter such attacks, cryptographers ensure that **RSA key sizes are sufficiently large** and use additional hardening techniques

Conclusion

The security of cryptographic systems depends on the strength of the underlying mathematical problems they rely on. However, as demonstrated in this document, various cryptanalysis techniques can effectively break or weaken encryption schemes when vulnerabilities exist.

- **The Vigenère Cipher**, though historically significant, is susceptible to **Kasiski Examination** and **Frequency Analysis**, making it insecure for modern applications.
- **The Meet-in-the-Middle (MITM) Attack** exposes the weakness of **2DES**, reducing its security level and reinforcing the necessity of using stronger encryption standards like **3DES and AES**.
- **Pollard's Rho Algorithm** provides an efficient method for **integer factorization**, which can be used to break cryptographic systems dependent on the difficulty of factorizing composite numbers, such as RSA with small key sizes.

These cryptanalytic methods emphasize the continuous need for innovation in cryptographic security. As computational power increases, new and more advanced cryptographic techniques must be developed to counter evolving threats. Understanding these attacks not only helps in identifying weaknesses in existing systems but also plays a crucial role in designing robust encryption methods that safeguard sensitive information in the digital world.