Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Математичні основи захисту інформації

Екзаменаційна робота

Виконав студент 3-го курсу
Групи ІПС-32
Ольховатий Ігор

Екзаменаційний білет №13

1. Проблема дискретного логарифма, часові характеристика складності цієї проблеми та шифр, який побудований на цій проблемі.

Проблема дискретного логарифма є однією з основних концепцій в криптографії. Вона полягає в знаходженні логарифму числа b за модулем простого числа p, коли відоме число a, таке що $a^x \equiv b \ (mod\ p)$. Ця задача вважається складною через те, що, на відміну від звичайного логарифмування, ефективний алгоритм для її розв'язання не відомий, особливо для великих чисел. Подібно, якщо g і h елементи зі скінченної циклічної групи G, тоді розв'язок x рівняння $g^x = h$ зветься дискретним логарифмом h за основою g в групі G.

Для довільної мультиплікативної групи для розв’язку задачі дискретного логарифмування в алгоритмі використовується таблиця, що складається з $O(\sqrt{|<g>|})$ пар елементів і виконується $O(\sqrt{|<g>|})$ множень. Цей алгоритм повільний і не придатний для практичного застосування.
Для конкретних груп існують ефективніші алгоритми, наприклад у кільці з лишків за певним модулем для розв’язку рівняння $a^x \equiv b \ (mod\ p)$, де p – просте і b не ділиться на p розв’язок можна знайти за $O(\sqrt{p} * \log p)$.

Одним з найвідоміших є шифрів, які базуються на дискретному л огарифмі є шифр Ель-Гамаля. Цей шифр є асиметричним, що означає використання двох ключів — відкритого та закритого. Безпека Ель-Гамаль шифрування заснована на складності обчислення дискретних логарифмів для певних (дуже ретельно підібраних дискретних груп), що робить його важким для злому без знання приватного ключа.

2. Знайти всі твірні мультиплікативної групи поля F13. Скільки має бути таких твірних?

Для розв’язання цієї задачі напишемо невелику програму мовою Python:

```
ef is_generator(element, field_size):
    if element == 0:
        return False
    for k in range(1, field_size - 1):  # Check for powers from 1 to
field_size-2
        if pow(element, k, field_size) == 1:
            return False
    return True
field_size = 13
generators = [element for element in range(1, field_size) if
is_generator(element, field_size)]
```

Дана програма перевіряє всі степені від 1 до (13-2) = 1 і елемент у цій степені дорівнює 1, то за допомогою нього можна буде утворити всі елементи групи, отже він є її генератором.

Твірними (генераторами) мультиплікативної групи поля $F13$ є елементи 2, 6, 7 та 11. Усього є 4 таких твірних.

3. Зашифрувати шифром Віженера повідомлення KUKURIKU за допомогою ключа RIKITI.

Шифр Віженера — поліалфавітний шифр, який як ключ використовує слово. Якщо пронумерувати літери алфавіту від 0 до 32 (а → 0, б → 1, в → 2, …), то шифрування Віженера можна подати формулою: Ci = (Pi + Kj) mod 33, де Kj — j-та літера ключового слова, Pi — i-а літера вихідного слова. Ключове слово повторюється, поки не отримано гаму, рівну довжині повідомлення. Дешифрування відбувається за наступною формулою: Ci = (Pi + 33 - Kj) mod 33.

Для шифрування повідомлення напишемо невелику програму мовою Python:

```
def extend_key_to_match_string_length(string, key):
    """Extend the key to match the length of the string."""
    key_list = list(key)
    if len(string) == len(key_list):
        return key_list
    else:
        for i in range(len(string) - len(key_list)):
            key_list.append(key_list[i % len(key)])
    return "".join(key_list)


def encrypt_text(text: str, key: str, alphabet_length: int = 26):
    """Encrypt the string using the key."""
    cipher_text = []
    for char, key_char in zip(text, key):
        x = (ord(char) + ord(key_char)) % alphabet_length
        x += ord('A')
        cipher_text.append(chr(x))
    return "".join(cipher_text)


def decrypt_text(text: str, key: str, alphabet_length: int = 26):
    """Decrypt the cipher text using the key."""
    original_text = []
    for cipher_char, key_char in zip(text, key):
        x = (ord(cipher_char) - ord(key_char) + alphabet_length) %
alphabet_length
        x += ord('A')
        original_text.append(chr(x))
    return "".join(original_text)
```

```
if __name__ == "__main__":
    input_string = "KUKURIKU"
    keyword = "RIKITI"
    key = extend_key_to_match_string_length(input_string, keyword)
    encrypted_text = encrypt_text(input_string, key)
    decrypted_text = decrypt_text(encrypted_text, key)
    print(f"Encrypted Text: {encrypted_text}")
    print(f"Original Text: {decrypted_text}")
```

Encrypted Text: BCUCKQBC

Original Text: KUKURIKU

Бачимо, що програма правильно зашифрувала та дешифрувала повідомлення.