Отчёт по лабораторной работе №3

Ван Яо

Содержание

Список таблиц

Список иллюстраций

Цель работы

1. Изучить теоретические основы шифрования гаммированием
2. Реализовать алгоритм шифрования с использованием конечной гаммы
3. Исследовать свойства и криптостойкость метода гаммирования
4. Проанализировать преимущества и недостатки различных подходов к генерации гаммы

Ход лабораторной работы

Теоретические основы

Основные понятия

Гаммирование — процедура наложения на исходный текст гаммы шифра (псевдослучайной последовательности) при помощи некоторой функции.

Однократное использование (one-time pad) — идеально безопасная схема шифрования, где длина ключа равна длине сообщения.

Псевдослучайная последовательность (ПСП) — последовательность, статистически неотличимая от случайной, но генерируемая детерминированным алгоритмом.

Математические основы

Операция сложения по модулю

Для шифрования используется операция сложения по модулю N:

$$c_i = (p_i + k_i) \mod N$$

где:

1. (p_i) — i-й символ открытого текста
2. (k_i) — i-й символ гаммы
3. (c_i) — i-й символ шифртекста
4. (N) — мощность алфавита

Для дешифрования:

$$p_i = (c_i - k_i) \mod N$$

Генератор псевдослучайных чисел

Линейный конгруэнтный генератор:

$$\gamma_i = a \cdot \gamma_{i-1} + b \mod m$$

где (a, $gamma_0$, b) — ключевые параметры.

Типы гаммирования

С конечной гаммой — используется повторяющаяся последовательность (фраза, слово)

С бесконечной гаммой — используется генератор ПСП

Однократное использование — истинно случайная гамма длиной, равной сообщению

Практическая реализация

Реализация шифрования конечной гаммой

```
    def encrypt(self, plaintext, gamma):

        plain_numbers = self.text_to_numbers(plaintext)
        gamma_numbers = self.text_to_numbers(gamma)

        if len(gamma_numbers) < len(plain_numbers):

            repeated_gamma = []
            while len(repeated_gamma) < len(plain_numbers):
                repeated_gamma.extend(gamma_numbers)
            gamma_numbers = repeated_gamma[:len(plain_numbers)]
        else:
            gamma_numbers = gamma_numbers[:len(plain_numbers)]

        cipher_numbers = []
        for p, g in zip(plain_numbers, gamma_numbers):
            cipher_num = (p + g) % self.modulus
            cipher_numbers.append(cipher_num)

        return self.numbers_to_text(cipher_numbers)
```

```
def decrypt(self, ciphertext, gamma):

    cipher_numbers = self.text_to_numbers(ciphertext)
    gamma_numbers = self.text_to_numbers(gamma)


    if len(gamma_numbers) < len(cipher_numbers):

        repeated_gamma = []
        while len(repeated_gamma) < len(cipher_numbers):
            repeated_gamma.extend(gamma_numbers)
        gamma_numbers = repeated_gamma[:len(cipher_numbers)]
    else:
        gamma_numbers = gamma_numbers[:len(cipher_numbers)]

    plain_numbers = []
    for c, g in zip(cipher_numbers, gamma_numbers):
        plain_num = (c - g) % self.modulus
        plain_numbers.append(plain_num)

    return self.numbers_to_text(plain_numbers)
```

Реализация линейного конгруэнтного генератора

```python
class LCGGenerator:

    def __init__(self, a, seed, b, m):

        self.a = a
        self.current = seed
        self.b = b
        self.m = m

    def next(self):

        self.current = (self.a * self.current + self.b) % self.m
        return self.current

    def generate_sequence(self, length):

        sequence = [self.current]
        for _ in range(length - 1):
            sequence.append(self.next())
        return sequence
```

Функциональное тестирование

Исходный текст	Гамма	Метод	Результат	Статус
“ПРИКАЗ”	“ГАММА”	Конечная гамма	“УСХЧБЛ”	√
“КРИПТОГРАФИЯ”	“КЛЮЧ”	Конечная гамма	“ШЭЬПЫИЦФДЧЛ”	√
“СЕКРЕТ”	seed=5, a=7, b=3, m=32	LCG	“ФЙМСЧО”	√

Анализ криптостойкости

Преимущества гаммирования

1. **Теоретическая стойкость** — при использовании одноразового ключа
2. **Высокая скорость** — побитовые операции выполняются быстро
3. **Простота реализации** — минимальные вычислительные ресурсы
4. **Отсутствие распространения ошибок** — ошибка в одном бите не влияет на другие

Недостатки и уязвимости

1. **Проблема распределения ключей** — необходимо безопасно передать длинный ключ
2. **Периодичность гаммы** — при использовании конечной гаммы
3. **Уязвимость к известному открытому тексту** — если известна пара (открытый текст, шифртекст)
4. **Статистические атаки** — при недостаточной случайности гаммы

Выводы

1. **Теоретические знания:** Изучены математические основы гаммирования, различные подходы к генерации псевдослучайных последовательностей и их криптографические свойства.
2. **Практические навыки:** Реализованы алгоритмы шифрования как с конечной гаммой, так и с использованием генераторов псевдослучайных чисел.
3. **Аналитические способности:** Проанализированы преимущества и недостатки различных схем гаммирования, выявлены основные уязвимости метода.
4. **Сравнительный анализ:** Показано, что безопасность гаммирования полностью зависит от свойств гаммы: ее случайности, длины и периодичности.
5. **Рекомендации:** Для обеспечения высокой безопасности следует использовать криптографически стойкие генераторы псевдослучайных чисел или, в идеале, схему однократного использования с истинно случайными ключами.

Литература

Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. М.: Гелиос АРВ, 2005.

Приложения

Полный исходный код программ доступен в репозитории GitHub: https://github.com/wangyao200036/cryptography-labs/tree/main/lab3