НАО «Восточно-Казахстанский технический университет имени Д. Серикбаева»

Дисциплина «Теоретико-числовые методы в криптографии»

САМОСТОЯТЕЛЬНЫЙ ПРАКТИКУМ К ЛЕКЦИИ 13

Линейные регистры над конечным полем и восстановление последовательностей

Методические указания и задания для самостоятельной работы

Статус: дополнительный практикум к лекции 13. В рабочей программе практическая тема 13 отмечена «Убрать?» и имеет 0 аудиторных часов. Материал предназначен для самостоятельного изучения и продолжает принятую структуру комплекта.

Цель: построить двоичный регистр, исследовать его период и статистические свойства, проверить примитивность многочлена и восстановить другую рекурсию по наблюдаемому фрагменту.

Средства: Python 3, без внешних библиотек. Все опыты проводятся на заданных учебных последовательностях степени от 3 до 6. Сначала изучите методику и примеры разделов 1–4, затем выполните свой вариант из раздела 5.

1 Соглашения и порядок исследования

Работа выполняется над $\mathbb{F}_2$. Сложение — XOR, а числа 0 и 1 являются элементами поля. Рекурсия записывается в виде

$$u_{t+m} = \sum_{j=0}^{m-1} c_j \, u_{t+j} \pmod 2.$$

Строка коэффициентов `1100` означает $c_0 = 1, c_1 = 1, c_2 = c_3 = 0$. Поэтому она задаёт $u_{t+4} = u_t + u_{t+1}$ и многочлен $X^4 + X + 1$. Строка начального состояния содержит $u_0, \dots, u_{m-1}$ слева направо. Выходной бит — **первый** разряд состояния до сдвига.

При каждом шаге запишите текущий выход, вычислите новый бит, удалите первый разряд и добавьте новый справа. Для определения периода отмечайте уже посещённые состояния. Повтор первого выходного бита ещё не означает повтор состояния.

Исследование состоит из пяти связанных действий: построение траектории; проверка многочлена; подсчёт статистики полного периода; учебное гаммирование; восстановление неизвестной рекурсии. Данные последнего действия приведены отдельно и относятся к другому регистру.

2 Методика и разобранные примеры

2.1 Ручное построение цикла

Пусть $c = (1,1,0)$ и $S_0 = 100$. Новый бит равен XOR первого и второго разрядов.

t	Состояние	Выход u_t	Новый бит	Следующее состояние
0	100	1	1	001
1	001	0	0	010
2	010	0	1	101
3	101	1	1	011
4	011	0	1	111
5	111	1	0	110
6	110	1	0	100

Получен период `1001011` длины 7, предпериод отсутствует. Нулевое состояние в цикл не входит. Изменение ненулевого начального состояния того же примитивного регистра меняет фазу, но сохраняет длину периода.

2.2 Проверка многочлена

Для $f = X^3 + X + 1$ отсутствие корней 0 и 1 доказывает неприводимость. Поскольку $2^3 - 1 = 7$ — простое число, ненулевой класс $X \bmod f$, отличный от 1, имеет порядок 7. Значит, период максимален.

Для $f = X^4 + X + 1$ сначала исключают линейные делители и делимость на $X^2 + X + 1$. Затем используют $M = 15 = 3 \cdot 5$. В факторполе выполнено

$$X^5 \equiv X^2 + X \neq 1, \qquad X^3 \not\equiv 1 \pmod f.$$

Эти проверки доказывают примитивность. Проверять все показатели от 1 до 15 для этого не требуется.

Контрпример. Для $g = X^4 + X^3 + X^2 + X + 1$ неприводимость сохраняется, но $X^5 \equiv 1 \pmod g$. Регистр с коэффициентами `1111` и состоянием `1000` имеет период `10001` длины 5. Поэтому «неприводимый» и «примитивный» не являются синонимами.

2.3 Баланс и периодическая корреляция

В периоде `1001011` три нуля и четыре единицы. Это соответствует формулам $N_0 = 2^{m-1} - 1$ и $N_1 = 2^{m-1}$ при $m = 3$.

Для корреляции замените биты знаками $a_t = 1 - 2u_t$ и вычислите

$$C(k) = \sum_{t=0}^{N-1} a_t \, a_{(t+k) \bmod N}.$$

Для рассматриваемого периода получается $C(0) = 7$ и $C(1) = \cdots = C(6) = -1$. Например, при $k = 1$ три пары знаков совпадают и четыре различаются, поэтому сумма равна $3 - 4 = -1$.

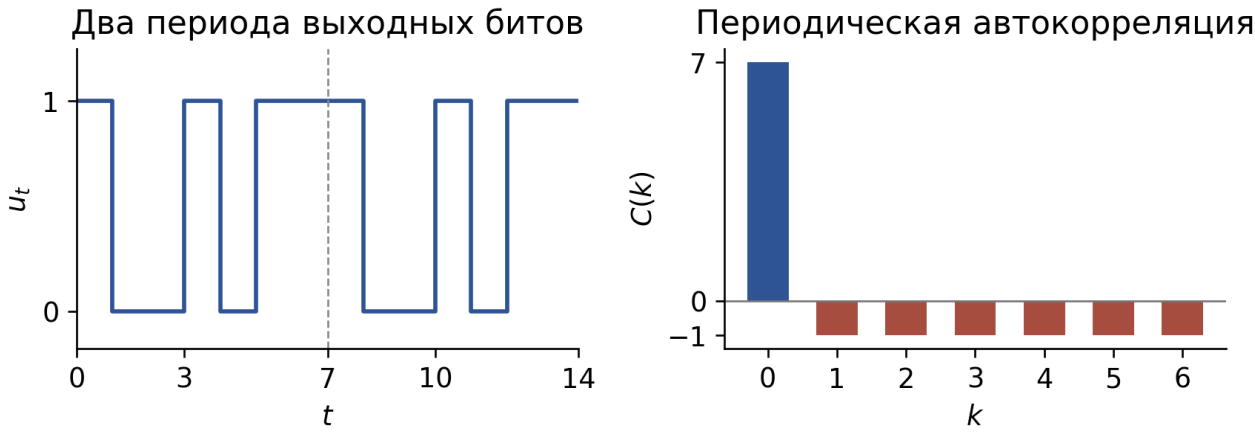


Рисунок 1 — Образец исследования полного периода

При вычислении используется циклическое продолжение периода. Если исходная траектория имеет предпериод, его сначала исключают. Для короткого произвольного фрагмента формулы m-последовательности не применяются.

2.4 Восстановление рекурсии по наблюдениям

Предположим, известна длина $d = 3$ неизвестной минимальной рекурсии и наблюдается строка `1001011100`. Первые $2d = 6$ битов используются для определения коэффициентов, остальные четыре — только для проверки.

Система по первым шести битам:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Матрица невырождена, откуда $c = (1,1,0)$. Начальное состояние равно `100`. Воспроизведение первых десяти битов даёт всю наблюдаемую строку, включая контрольный хвост `1100`. Следующие восемь битов **после всей строки из десяти битов** равны `10111001`.

Для строки `111111` при $d = 3$ все строки матрицы одинаковы и все уравнения сводятся к $c_0 + c_1 + c_2 = 1$. Ранг равен 1, коэффициенты неоднозначны. Сама постоянная последовательность имеет минимальный порядок 1. Для фрагмента `0010` при $d = 2$ первое уравнение имеет вид $0 = 1$, поэтому система несовместна.

2.5 Учебное гаммирование

Первые восемь битов регистра с $c = 110$, $S_0 = 100$ равны `10010111`. Для открытого текста `11001010` получаем

$$11001010 \oplus 10010111 = 01011101.$$

Повторное сложение шифртекста с той же гаммой возвращает исходную строку. Эта обратимость не означает стойкости: по известным битам текста и шифртекста гамма восстанавливается, а её линейная рекурсия предсказывается методом предыдущего подраздела.

3 Реализация на Python

Сохраните блоки кода в одном файле в указанном порядке. Начальные данные удобно задавать строками и преобразовывать функцией `bits`. Внутренние функции `step`, `poly_rem`, `poly_mul`, `x_power` и `solve_gf2` вызываются из основных функций с согласованными параметрами.

3.1 Входные данные и генерация

```
def bits(text):
    if not text or any(x not in "01" for x in text):
        raise ValueError("Нужна непустая строка из 0 и 1")
    return tuple(map(int, text))

def check_register(c, seed):
    if not c or len(c) != len(seed):
        raise ValueError("Не совпадают длины")
    if any(x not in (0, 1) for x in tuple(c) + tuple(seed)):
        raise ValueError("Коэффициенты и состояние должны быть битами")
def step(state, c):
    new = sum(x * y for x, y in zip(state, c)) % 2
    return state[1:] + (new,)

def generate(c, seed, count):
    check_register(c, seed)
    if count < 0:
        raise ValueError("Отрицательная длина")
    state, out = tuple(seed), []
    for _ in range(count):
        out.append(state[0])
        state = step(state, c)
    return out
```

`generate` выдаёт ровно `count` битов, включая биты начального состояния. Это соглашение используется во всех таблицах и ответах. Например, при состоянии `100` первые три бита выхода — именно `100`.

3.2 Определение предпериода и периода

```
def orbit(c, seed):
    check_register(c, seed)
    state, seen, states, out = tuple(seed), {}, [], []
    while state not in seen:
        seen[state] = len(states)
        states.append(state)
        out.append(state[0])
        state = step(state, c)
    mu = seen[state]
    period = len(states) - mu
    return mu, period, out, states
```

Возвращаются длина предпериода `mu`, длина цикла `period`, выход `out` до первого повтора состояния и список посещённых состояний. Если `mu` ненулевое, полный период выхода находится в `out[mu:]`; предпериод — в `out[:mu]`.

3.3 Статистика полного периода

```
def statistics(word):
    if not word or any(x not in (0, 1) for x in word):
        raise ValueError("Нужен непустой двоичный период")
    N = len(word)
    signs = [1 - 2 * x for x in word]
    corr = [sum(signs[t] * signs[(t + k) % N]
                for t in range(N)) for k in range(N)]
    return N - sum(word), sum(word), corr
```

Результат — число нулей, число единиц и список $C(0), \dots, C(N-1)$. Вычисления целочисленные и не используют нормирование. Проверяйте именно полный период, найденный по состояниям.

3.4 Многочлены в двоичном представлении

Многочлен $a_0 + a_1 X + \cdots + a_m X^m$ кодируется целым числом $\sum a_j 2^j$. Например, $X^3 + X + 1$ кодируется числом 11 с двоичной записью `1011`. Это запись многочлена по степеням; она отличается от строки коэффициентов рекурсии `110`.

Сложение многочленов выполняется XOR, умножение на X — сдвигом влево. В `poly_rem` вычитается сдвинутая копия делителя, пока степень остатка не станет меньше степени делителя. Функция `x_power` вычисляет остаток X^k для неотрицательного показателя k.

```
def poly_rem(a, f):
    if f <= 0:
        raise ValueError("Нулевой делитель многочленов")
    while a and a.bit_length() >= f.bit_length():
        a ^= f << (a.bit_length() - f.bit_length())
    return a

def poly_mul(a, b, f):
    out = 0
    while b:
        if b & 1:
            out ^= a
        a <<= 1
        b >>= 1
    return poly_rem(out, f)

def x_power(k, f):
    out, x = 1, 2
    while k:
        if k & 1:
            out = poly_mul(out, x, f)
        x = poly_mul(x, x, f)
        k >>= 1
    return out
```

3.5 Неприводимость и примитивность

Многочлен степени m приводим тогда и только тогда, когда имеет монический делитель степени от 1 до $\lfloor m/2 \rfloor$. Для учебных степеней до 6 такой перебор мал. Функция `irreducible` проверяет именно делимость многочленов над $\mathbb{F}_2$, а не целочисленную делимость их кодов.

```
def prime_divisors(n):
    out, d = [], 2
    while d * d <= n:
        if n % d == 0:
            out.append(d)
            while n % d == 0:
                n //= d
        d += 1
    return out + ([n] if n > 1 else [])

def irreducible(f):
    m = f.bit_length() - 1
    if m < 1:
        return False
    for d in range(1, m // 2 + 1):
        for g in range(1 << d, 1 << (d + 1)):
            if poly_rem(f, g) == 0:
                return False
    return True
```

Следующая функция строит многочлен из коэффициентов и вычисляет остатки, необходимые для критерия примитивности:

```
def certificate(c):
    check_register(c, c)
    m = len(c)
    f = (1 << m) + sum(x << j for j, x in enumerate(c))
    M = (1 << m) - 1
    irred = irreducible(f)
    checks = [(r, x_power(M // r, f))
              for r in prime_divisors(M)]
    primitive = (c[0] == 1 and irred
                 and x_power(M, f) == 1
                 and all(v != 1 for r, v in checks))
    return primitive, f, checks
```

Возвращаются логический результат, целочисленный код f и пары `(r, остаток)`. Каждая пара относится к степени $X^{M/r}$, где $M = 2^m - 1$. Остатки возвращаются в том же двоичном кодировании многочленов. Неприводимость проверяется до окончательного вывода.

3.6 Решение системы над двоичным полем

Ведущий элемент над $\mathbb{F}_2$ всегда равен 1, поэтому деление строки на него не требуется. Для исключения единицы в другом месте прибавляется ведущая строка операцией XOR.

```
def solve_gf2(A, b):
    rows = [list(row) + [y] for row, y in zip(A, b)]
    n, rank, pivots = len(A[0]), 0, []
    for col in range(n):
        pivot = next((i for i in range(rank, len(rows))
                      if rows[i][col]), None)
        if pivot is None:
            continue
        rows[rank], rows[pivot] = rows[pivot], rows[rank]
        for i in range(len(rows)):
            if i != rank and rows[i][col]:
                rows[i] = [x ^ y for x, y in zip(rows[i], rows[rank])]
        pivots.append(col)
        rank += 1
    if any(not any(row[:n]) and row[n] for row in rows):
        raise ValueError("Система несовместна")
    if rank < n:
        raise ValueError("Коэффициенты определены неоднозначно")
    out = [0] * n
    for i, col in enumerate(pivots):
        out[col] = rows[i][n]
    return out
```

Программа различает несовместную систему и систему с несколькими решениями. При полном ранге она возвращает коэффициенты в порядке $c_0, \dots, c_{d-1}$.

3.7 Восстановление и проверка контрольного хвоста

```
def recover(prefix, m):
    if m < 1 or len(prefix) < 2 * m:
        raise ValueError("Недостаточно данных")
    if any(x not in (0, 1) for x in prefix):
        raise ValueError("Ожидаются биты")
    H = [prefix[t:t + m] for t in range(m)]
    c = solve_gf2(H, prefix[m:2 * m])
    predicted = generate(c, prefix[:m], len(prefix))
    if predicted != list(prefix):
        raise ValueError("Не совпадают контрольные биты")
    return c
```

Коэффициенты находятся по первым $2d$ битам. Если передан более длинный фрагмент, оставшиеся биты проверяются после решения. Несовпадение означает, что полученная рекурсия не описывает все наблюдения.

4 Запуск примера и проверка результата

```
c, seed = bits("110"), bits("100")
mu, N, word, states = orbit(c, seed)
print("Предпериод и период:", mu, N)
print("Один период:", "".join(map(str, word)))
print("Нули, единицы, корреляция:", statistics(word))
print("Сертификат:", certificate(c))
observed = generate(c, seed, 10)
found = recover(observed, 3)
print("Восстановленные коэффициенты:", found)
print("Следующие 8 бит:", generate(found, observed[:3], 18)[10:])
```

Ожидаемый вывод

```
Предпериод и период: 0 7
Один период: 1001011
Нули, единицы, корреляция: (3, 4, [7, -1, -1, -1, -1, -1, -1])
Сертификат: (True, 11, [(7, 2)])
Восстановленные коэффициенты: [1, 1, 0]
Следующие 8 бит: [1, 0, 1, 1, 1, 0, 0, 1]
```

В строке сертификата число 11 означает многочлен $X^3 + X + 1$, а пара `(7, 2)` — остаток $X^{7/7} = X$, закодированный числом 2. Этот остаток не равен 1. Все операции выполняются в конечном поле; вещественная арифметика и округление не нужны.

5 Индивидуальные задания

Таблица 1 задаёт регистр для исследования и 16-битный открытый текст. **Таблица 2** содержит наблюдение другого регистра: его минимальная длина d известна, коэффициенты требуется найти. В обоих заданиях номер варианта одинаков.

Задание 1. Рекурсия и цикл. По таблице 1 запишите рекурсию и многочлен f. Выполните вручную первые $m + 2$ перехода с указанием состояния, выходного и нового бита. Программно найдите предпериод, период и все различные состояния. Приведите выход за один полный период.

Задание 2. Примитивность. Проверьте неприводимость f и разложите $M = 2^m - 1$ на простые множители. Вычислите остатки $X^{M/r} \bmod f$ для каждого различного простого делителя r. Запишите остатки как многочлены или как двоичные коды с явным указанием способа записи. Сопоставьте алгебраический вывод с найденным периодом. Деления при проверке неприводимости можно выполнить программно, но критерий необходимо объяснить.

Задание 3. Статистические свойства. Для полного периода подсчитайте N_0, N_1 и все $C(k)$. Постройте график выходных битов на двух периодах и график периодической автокорреляции любым доступным средством. Сопоставьте результаты с формулами m-последовательности. Укажите, почему эти свойства не доказывают криптографическую стойкость.

Задание 4. Гаммирование. Возьмите первые 16 битов выхода своего регистра и зашифруйте открытый текст из таблицы 1 операцией XOR. Проверьте расшифрование. Восстановите эти 16 битов гаммы по открытой и зашифрованной строкам и объясните, какую информацию это даёт для линейной модели.

Задание 5. Неизвестная рекурсия. Используя таблицу 2, постройте матрицу $d \times d$ по первым $2d$ битам и правую часть. Решите систему над $\mathbb{F}_2$, найдите коэффициенты и начальное состояние. Проверьте четыре контрольных бита в конце наблюдения. Предскажите следующие восемь битов **после всего наблюдаемого фрагмента длины** $2d + 4$. Коэффициенты из таблицы 1 в этом задании не используются.

Задание 6. Особые случаи. Исследуйте нулевое состояние своего регистра. Для `c=1111`, `seed=1000` сравните неприводимость, примитивность и период. Для `c=010`, `seed=100` найдите предпериод и цикл. Объясните результаты `recover(bits("111111"), 3)` и `recover(bits("0010"), 2)`. Наконец, измените последний контрольный бит строки из таблицы 2 и проверьте, что восстановление обнаруживает несовпадение.

Таблица 1 — Регистр и открытый текст для заданий 1–4

№	m	Коэффициенты	Состояние	Открытый текст
1	3	110	110	1010100001011010
2	3	101	110	1010101001010111
3	4	1100	1100	1010110001010100
4	4	1001	0100	1010111001010001
5	5	10100	01010	1011000001001110
6	5	10010	10001	1011001001001011
7	5	11110	11000	1011010001001000
8	5	11101	11111	1011011001000101
9	5	11011	00111	1011100001000010
10	5	10111	01110	1011101000111111
11	5	10100	10101	1011110000111100
12	5	10010	11100	1011111000111001
13	6	110000	100010	1100000000110110
14	6	110110	101001	1100001000110011
15	6	100001	110000	1100010000110000
16	6	111001	110111	1100011000101101
17	6	101101	111110	1100100000101010
18	6	110011	000110	1100101000100111
19	6	110000	001101	1100110000100100
20	6	110110	010100	1100111000100001

Таблица 2 — Наблюдение неизвестного регистра для задания 5

№	d	Наблюдаемый фрагмент
1	4	100100011110
2	5	01110000110101
3	6	0100111100000110
4	3	0111001011
5	4	111010110010
6	5	00011101111100
7	6	1001110110000011

№	d	Наблюдаемый фрагмент
8	3	0101110010
9	4	010001111010
10	5	10111010100001
11	6	1110111010110100
12	3	0010111001
13	4	100100011110
14	5	01100111000011
15	6	0100001011001010
16	3	1110010111
17	4	111010110010
18	5	00001011010100
19	6	1001001010100110
20	3	1100101110

6 Содержание отчёта и оценивание

Укажите дисциплину, тему, Ф. И. О., группу и вариант. Приведите исходные данные, ручную таблицу первых переходов, результаты программного исследования, объяснение критерия примитивности и графики. Для обратной задачи включите матрицу, правую часть, найденные коэффициенты и прогноз. Завершите отчёт выводом о различии статистических свойств и непредсказуемости.

Элемент работы	Баллы
Задание 1 — рекурсия, ручные переходы и период	20
Задание 2 — неприводимость и проверка примитивности	20
Задание 3 — баланс, корреляция и графики	15
Задание 4 — гаммирование и обратная проверка	10
Задание 5 — система, восстановление и прогноз	25
Задание 6 — объяснение особых случаев	10
Всего	100

Типичные ошибки: обратный порядок коэффициентов; выдача нового бита вместо первого разряда без изменения соглашения; определение периода по одному повторившемуся биту; смешение числового кода многочлена и обычного целого; вычисление нециклической корреляции; прогноз от конца обучающих битов вместо конца всего наблюдения; вывод о стойкости только по максимальному периоду.

7 Контрольные вопросы

1. Как строка `1100` преобразуется в рекурсию и многочлен?
2. Почему в определении периода нужно отслеживать полное состояние?
3. Что содержат `out[:mu]` и `out[mu:]` в результате `orbit`?

4. Почему нулевое состояние имеет период 1 при любом линейном законе?
5. Чем код многочлена `1011` отличается от строки коэффициентов `110`?
6. Почему неприводимость нельзя проверять обычным целочисленным остатком?
7. Какие степени нужно проверить для максимального периода 63?
8. Почему единиц в полном периоде m-последовательности на одну больше, чем нулей?
9. Чем ненормированная периодическая корреляция отличается от нормированной?
10. Как отличить несовместность системы от неоднозначности коэффициентов?
11. Для чего после первых $2d$ наблюдаемых битов оставлены четыре контрольных?
12. Почему успешно восстановленная линейная рекурсия позволяет вычислить дальнейшую гамму?

Литература

1. Лекция 13 данного комплекта — определения, доказательства периода, баланса, корреляции и восстанавливаемости.
2. Paar C., Pelzl J. **Understanding Cryptography.** 2010. § 2.3.1–2.3.2.
3. Trappe W., Washington L. C. **Introduction to Cryptography with Coding Theory.** 2nd ed., 2006. § 2.11.
4. Katz J., Lindell Y. **Introduction to Modern Cryptography.** 2nd ed., 2015. § 3.3.
5. Barker E. et al. **NIST SP 800-90C.** 2025. Конструкции генераторов случайных битов, для сопоставления учебной модели с криптографическими генераторами.