Математические основы криптологии.
Лабораторные работы.

1. Шифр Полибия (вводная).
2. Шифрование файлов. Четыре криптопримитива.
3. Многораундовое шифрование (требуется 2).
4. Криптографические тесты (NIST).
5. Хэш-функция. Электронная подпись (требуется 3).
6. Генератор псевдослучайных последовательностей (требуется 4).
7. Генерация ключей (требуется 5,6).
8. Арифметика с длинными целыми.

Курсовая работа (требуется 4,7).

Результатом выполнения каждой лабораторной работы должна быть написанная и отлаженная программа, реализующая поставленную в работе задачу. Допускается написание программ на любом языке программирования. Не рекомендуется использовать интерпретируемые языки, обладающие низкой вычислительной производительностью. Не рекомендуется менять язык программирования в процессе выполнения курса работ, так как модули и функции предыдущих работ используются в последующих. Рекомендуется язык C/C++.

Лабораторная работа №1 (вводная).
Шифр Полибия.

Цель работы: вспомнить основы работы с текстовыми файлами.

Постановка задачи: имеется текстовый файл длиной L символов. Создать квадратную таблицу (массив) размером $m \times n$, где $n = [\sqrt{L}] + 1$, $m = n$ или $m = n - 1$ (здесь квадратными скобками обозначается взятие целой части числа). Таблица загружается символами из входного файла построчно – сначала полностью заполняется первая строка, затем вторая и т. д. После этого текст из таблицы выгружается в выходной файл по столбцам – сначала полностью записывается первый столбец, затем второй и т. д.

Числа m и n следует подбирать таким образом, чтобы степень заполнения таблицы была максимальной. Допускаются и другие способы заполнения и считывания таблицы – например, по спирали. Рассмотреть случай, когда входной файл имеет очень большой размер и не может быть загружен в память весь сразу. Реализовать в программе функцию шифрования и функцию дешифрования текста.

Лабораторная работа №2.
Шифрование файлов. Четыре криптопримитива.

Цель работы: реализовать четыре основных криптографических примитива.

Постановка задачи: написать программу, осуществляющую шифрование и дешифрование произвольных файлов при помощи четырех примитивных криптоалгоритмов. Допускается как реализация всех криптоалгоритмов в разных программах, так и в одной программе, запускаемой в разных режимах. Программа должна генерировать ключ, сохранять его в ключевом файле, считывать ключ из ключевого файла, шифровать данный файл по ключу и расшифровывать закрытый файл.

Криптографический примитив – это несложный по времени и оперативной памяти обратимый алгоритм шифрования. На основе композиций криптографических примитивов можно создавать стойкие криптоалгоритмы.

1. Простая подстановка. Каждый байт из входного файла заменяется на другой байт согласно ключу. (Одинаковые байты будут заменяться одинаковыми значениями.) Ключом является последовательность байтов от 0 до 255, перемешанная случайным образом. В ключе каждое значение встречается ровно один раз. Для расшифровки удобно сгенерировать ключ обратной подстановки.

2. Перестановка (транспозиция). Из входного файла считывается блок данных, равный длине ключа. Байты в блоке переставляются согласно ключу. Перемешанный блок записывается в выходной файл. Ключом является последовательность чисел от 1 до N, перемешанная случайным образом, где N – длина блока. В ключе каждое значение встречается ровно один раз. Пример: если ключ 642351, а блок “ABCDEF”, то результатом перестановки будет “FDBCEA”. Для расшифровки удобно сгенерировать ключ обратной перестановки.

Рассмотреть случай, когда размер файла не кратен длине ключа. В этом случае при шифровании оставшийся «хвостик» можно дополнить до целого блока, а при дешифровании – отбросить лишние значения. Другой способ – сгенерировать по исходному ключу другой ключ меньшей длины.

3. Гаммирование (шифр Виженера). Из входного файла считывается блок данных, равный длине ключа. На каждый байт блока накладывается соответствующий байт ключа (гаммы) при помощи операции «исключающее ИЛИ» (xor, ^) или сложения (+). Результат наложения записывается в выходной файл. Ключом является случайная последовательность байт или символов произвольной длины. Для расшифровки файла выполняем наложение того же ключа еще раз операцией «xor» или вычитанием соответственно.

4. Одноразовый блокнот (шифр Вернама). Этот метод аналогичен гаммированию, но ключом является случайная последовательность байт, длиной равная размеру входного файла. Одноразовый блокнот лишь с большой натяжкой можно отнести к криптографическим примитивам. Создание истинно случайной последовательности ключа – операция чрезвычайно дорогая по времени и размерам оперативной памяти.

Метод проверки: файл произвольной структуры длиной несколько мегабайт сначала шифруется одним из криптоалгоритмов, затем дешифруется. Результат дешифрования побайтно сравнивается с исходным файлом (например, программой fc). Если файлы совпадают, то проверка пройдена. Проверка проводится для всех четырех криптопримитивов.

Лабораторная работа №3.
Многораундовое шифрование.

Цель работы: использовать криптографические примитивы для построения алгоритма многораундового шифрования.

Постановка задачи: написать программу, осуществляющую многораундовое шифрование и дешифрование произвольных файлов при помощи четырех примитивных криптоалгоритмов. Программа должна генерировать составной ключ, сохранять его в ключевом файле, считывать ключ из ключевого файла, шифровать данный файл по ключу и расшифровывать закрытый файл.

Раунд – это одно применение одного криптоалгоритма к одному блоку данных.

Ключом является последовательность раундов шифрования и набор ключей для отдельных криптоалгоритмов. Последовательность раундов генерируется случайным образом во время создания ключа. Количество раундов должно быть не менее 10. Ни один криптоалгоритм в последовательности не должен повторяться два раза подряд (чередования допустимы).

Программа считывает из входного файла блок данных, последовательно шифрует его криптоалгоритмами в соответствии с ключом, затем записывает результат в выходной файл. При расшифровке криптоалгоритмы применяются в обратном порядке. Длины блоков удобно выбирать равными степени двойки.

В качестве четвертого криптоалгоритма вместо одноразового блокнота следует использовать метод циклического побитового сдвига. Каждый байт (или 32-битное слово) из входного файла подвергается циклическому побитовому сдвигу влево (или вправо). Первый байт (слово) сдвигается на количество бит, равное первому элементу ключа, второе – равное второму, и т.д. Ключом является последовательность чисел от 0 до 7 в случае байтов (от 0 до 31 в случае 32-битных слов) произвольной длины. Для расшифровки выполняется сдвиг в другую сторону.

Для реализации циклического побитового сдвига можно использовать:

1. Инструкции процессора x86 ROR и ROL (вставка на языке ассемблера).
2. Функции группы _rotl(), _rotr() (Microsoft Visual C++).
3. Сочетание операций арифметического побитового сдвига (shl, shr, <<, >>) и побитового «или» (or, |). Пример: циклический сдвиг байта на 3 бита влево.
   11001010 rol 3 = 01010110
   11001010 << 3 = 01010000
   11001010 >> 5 = 00000110
   01010000 | 00000110 = 01010110

Метод проверки: файл произвольной структуры длиной несколько мегабайт сначала шифруется программой многораундового шифрования, затем дешифруется. Результат дешифрования побайтно сравнивается с исходным файлом (например, программой fc). Если файлы совпадают, то проверка пройдена.

Лабораторная работа №4.
Криптографические тесты (NIST).

Цель работы: разработать инструмент для получения криптографических характеристик произвольного файла.

Постановка задачи: написать программу, проводящую несколько криптографических тестов (взятых из стандарта NIST) над произвольным файлом. Результаты этих тестов показывают, насколько содержимое данного файла похоже на случайную последовательность байт. Результаты работы программы сохраняются в файле.

Будем называть файл «хорошим», если он по своим характеристикам мало отличим от случайной последовательности.

1. Частотные тесты.
   1) Подсчет количества бит n_0 и n_1 со значениями 0 и 1 соответственно.

      Расчет частот $p_0 = \frac{n_0}{n_{бит}}$, $p_1 = \frac{n_1}{n_{бит}}$, где $n_{бит}$ – длина файла в битах.

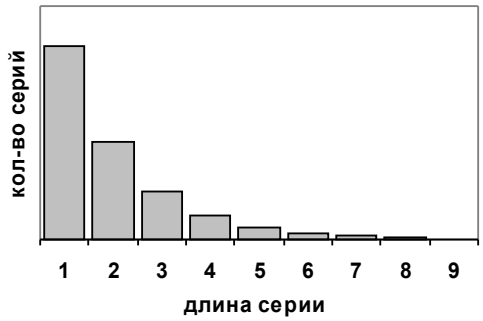
      Должно всегда выполняться $p_0 + p_1 = 1$, $0 \le p_0 \le 1$, $0 \le p_1 \le 1$. На «хороших» файлах $p_0 \approx p_1 \approx 0{,}5$ (равномерное распределение).
   2) Подсчет количества байт n_i со всеми возможными значениями $i \in [0;255]$.

      Расчет частот $p_i = \frac{n_i}{n_{байт}}$, где $n_{байт}$ – длина файла в байтах.

      Должно всегда выполняться $\sum_{i=0}^{255} p_i = 1$, $0 \le p_i \le 1 \ \ \forall i \in [0;255]$. На «хороших» файлах все p_i должны быть приближенно равны (равномерное распределение).

2. Энтропийные тесты. Здесь используются частоты, рассчитанные в п.1. Для вычисления логарифма по основанию 2 можно воспользоваться формулой $\log_b a = \frac{\log_c a}{\log_c b}$.
   1) Расчет энтропии бит $A_{бит} = -p_0 \log_2 p_0 - p_1 \log_2 p_1$. Должно всегда выполняться $0 \le A_{бит} \le 1$. На «хороших» файлах $A_{бит}$ должно стремиться к 1.
   2) Расчет энтропии байт $A_{байт} = -\sum_{i=0}^{255} p_i \log_2 p_i$. Слагаемые с $p_i = 0$ следует пропускать, т.к. $\log_2 0$ не существует. Должно всегда выполняться $0 \le A_{байт} \le 8$. На «хороших» файлах $A_{байт}$ должно стремиться к 8.

3. Серийные тесты. Файл рассматривается как последовательность битов.
   Серия – это последовательность идущих подряд битов с одинаковыми значениями (0 или 1).
   1) Подсчет количеств серий n_i одной и той же длины $i \in \mathbb{N}$. На «хороших» файлах количества серий должны соответствовать нормальному распределению (каждый следующий столбец гистограммы примерно вдвое меньше предыдущего).



2) Определение максимальной длины серии $i_{\max}$.

Должно всегда выполняться $\sum_{i=1}^{i_{\max}} n_i i = n_{бит}$, где $n_{бит}$ – длина файла в битах.

Пример: 1001111001000100101100000100001010010110010111010000111101000100
Для данной последовательности бит $i_{\max} = 5$, а длины серий приведены в таблице:

i, длина серии	n_i, кол-во серий
1	18
2	8
3	3
4	4
5	1

4. Матрично-ранговый тест. Из входного файла считывается блок длиной 1024 бита. Считанными битами заполняется матрица размером 32×32. Определяется, является эта матрица вырожденной или нет. (Матрица является *вырожденной*, если ее определитель равен нулю.) Подсчитывается количество вырожденных и невырожденных матриц. Процесс повторяется, пока не будет обработаны все блоки файла. Если длина файла в битах не кратна 1024, то последний неполный блок следует отбросить. На «хороших» файлах количество вырожденных матриц должно стремиться к нулю.
   Алгоритмическая сложность вычисления определителя матрицы $n \times n$ является экспоненциальной. Для ускорения работы алгоритма предлагается использовать метод Гаусса приведения к верхнетреугольному виду (см. приложение №1). Этот метод хотя и не получает значение определителя явно, однако позволяет оценить, вырождена матрица или нет.
   В начале работы метода каждым элементом матрицы является 0 или 1. В процессе работы алгоритма могут получаться числа как больше единицы, так и меньше нуля, а также дробные числа. Поэтому в качестве элементов матрицы следует использовать действительные числа (real, float, double). При сравнении действительных чисел с нулем нужно учесть погрешность вычислений.
5. Тест на сжимаемость. Провести компрессию файла при помощи LZ-алгоритма. Допускается использовать внешние библиотеки или программы (zip, 7z, Rar и проч.). Рассчитать коэффициент сжатия $k = \frac{n_{сжат}}{n_{исх}}$, где $n_{исх}$ и $n_{сжат}$ – длины исходного и сжатого файлов в байтах соответственно. На «хороших» файлах k должен стремиться к 1 или даже незначительно превосходить 1.

Метод проверки: написанная программа проверяется на текстовом файле (высокая избыточность), сжатом файле (zip-архив или jpg-изображение, низкая избыточность) и файле, содержащем псевдослучайную последовательность (избыточность отсутствует). Результаты работы программы сравниваются с эталонными.

Лабораторная работа №5.
Хэш-функция. Электронная подпись.

Цель работы: применить многораундовое шифрование для построения хэш-функции, использовать хэш-функцию для работы с электронной подписью.

Постановка задачи: добавить в программу многораундового шифрования (лабораторная работа №3) проверку целостности файла, используя электронную подпись. При шифровании файла программа должна вычислить его электронную подпись и записать ее в конец файла. При дешифровании файла программа должна снова вычислить электронную подпись по уже расшифрованным данным и сравнить ее с подписью, ранее записанной в файле. В зависимости от результата сравнения уведомить пользователя, был ли файл (или ключ) поврежден.

Пусть p_i – очередной блок данных, считанный из файла, H_i – очередное значение хэш-функции, H_{i-1} – предыдущее значение хэш-функции, $E_k(p)$ – функция многораундового шифрования блока данных p по составному ключу k. Здесь p_i и H_i – это массивы байтов одинаковой длины. Тогда *хэш-функцию* можно определить как

$$H_i = E_k(p_i \oplus H_{i-1}) \oplus p_i \quad \forall i = 1, 2, 3, \ldots , \quad H_0 = \text{const},$$

где знаком $\oplus$ обозначена операция «исключающее ИЛИ» (xor, ^). На считанный из файла блок данных накладывается предыдущее значение хэш-функции при помощи побитовой операции «xor». Результат наложения шифруется при помощи функции многораундового шифрования. На результат шифрования снова накладывается исходный блок данных. Результат этих преобразований и становится новым значением хэш-функции.

Начальное значение H_0 можно заполнить нулями или произвольными значениями (но всегда одинаковыми). В том случае, если длина файла не кратна длине блока, последний блок p_n дополняется до целого нулями или произвольными значениями (но всегда одинаковыми). Последнее значение хэш-функции H_n и будет *электронной подписью* для всего файла.

Метод проверки: файл произвольной структуры длиной несколько мегабайт шифруется программой с созданием электронной подписи.

1. Проводится контрольное дешифрование – программа должна сообщить, что повреждений нет.
2. Затем в зашифрованном файле изменяется один байт, и снова проводится дешифрование. Программа должна сообщить о наличии повреждения.
3. Зашифрованный файл восстанавливается, в ключевом файле изменяется один байт. Проводится дешифрование; программа должна снова сообщить о наличии повреждения.

Лабораторная работа №6.
Генератор псевдослучайных последовательностей.

Цель работы: разработать генератор псевдослучайных последовательностей большой разрядности.

Постановка задачи: реализовать генератор псевдослучайных последовательностей в виде библиотеки функций (классов). Написать программу, использующую этот генератор для создания файла с псевдослучайной последовательностью длиной в несколько мегабайт. Провести проверку созданного файла при помощи программы криптографических тестов (лабораторная работа №4).

Псевдослучайные значения, формируемые генератором, через некоторое число итераций всегда начинают повторяться. Это число называется *периодом* генератора (T). Постараемся построить генератор с максимальным периодом для данной разрядности n.

Пусть A – матрица $n \times n$, где $n = 2^k$, $n \geq 64$. Матрица A заполняется нулями и единицами следующим образом:

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_{n-2} & a_{n-1} & a_n \\ 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \end{pmatrix},$$

где $a_1, \dots, a_n$ – коэффициенты примитивного многочлена над полем вычетов по модулю 2 ($GF(2)$) вида $a_n x^n + a_{n-1} x^{n-1} + \dots + a_3 x^3 + a_2 x^2 + a_1 x + a_0$ и $a_i, x \in \{0;1\}$. В этом случае период генератора будет максимально возможным: $T = 2^n - 1$. Примитивный многочлен можно взять из [1] или подобрать самостоятельно. Состояние генератора описывается вектором v, состоящим из n элементов (нулей и единиц – битов). Начальное состояние генератора называется *зерном* и обозначается v_0. Зерно может быть произвольным ненулевым вектором.

При умножении матрицы A на вектор v получается новый вектор, элементы которого сдвинуты на один и добавлен один новый псевдослучайный элемент. Чтобы полностью заполнить вектор n новыми элементами, надо использовать матрицу A^n. Переход от одного состояния генератора к другому происходит так:

$$v_{i+1} = A^n v_i \quad \forall i = 0, 1, 2, \dots$$

Для данного многочлена матрицу A^n можно вычислить только один раз и сохранить в файле для последующего использования. Рекомендуется воспользоваться алгоритмом быстрого возведения в степень, сложность которого $O(\log_2 n)$.

Все операции с векторами и матрицами проводятся над $GF(2)$. При всех операциях над его элементами (нулями и единицами) должны получаться только эти же элементы. Для умножения элементов можно использовать операцию «И» (and, &), а для сложения – операцию «исключающее ИЛИ» (xor, ^).

Метод проверки: создается файл с псевдослучайной последовательностью. Этот файл проверяется при помощи криптографических тестов. Результаты тестов должны совпадать или быть весьма близкими с результатами для случайной последовательности.

Лабораторная работа №7.
Генерация ключей.

Цель работы: применить генератор псевдослучайных последовательностей большой разрядности для генерации стойких ключей.

Постановка задачи: добавить в программу многораундового шифрования с подписью (лабораторная работа №5) свой генератор псевдослучайных последовательностей (лабораторная работа №6) вместо стандартного для генерации составного ключа. Реализовать два способа:

1. Создание ключа по счетчику. Для инициализации зерна генератора используется значение счетчика мили- или микросекунд аппаратных часов компьютера либо значение счетчика тактов процессора (инструкция RDTSC). Созданный ключ сохраняется в файле.
2. Создание ключа по паролю. Для инициализации зерна генератора используется вводимая пользователем символьная строка произвольной длины. Рекомендуется учитывать только 4–5 младших битов от каждого символа, так как старшие биты могут совпадать. Одна и та же фраза должна приводить к генерации одного и того же ключа. Созданный ключ в файле не сохраняется.

Метод проверки:

1. Создается ключ по счетчику, выполняется шифрование файла длиной несколько мегабайт. Выполняется дешифрование файла по тому же ключу. Результат дешифрования побайтно сравнивается с исходным файлом (например, программой fc).
2. Создается ключ по паролю, выполняется шифрование файла длиной несколько мегабайт. Выполняется дешифрование файла с пересозданием ключа по тому же паролю. Результат дешифрования побайтно сравнивается с исходным файлом.

Если в обоих случаях файлы совпадают, то проверка пройдена.

Лабораторная работа №8.
Арифметика с длинными целыми.

Цель работы: создание библиотеки операций для работы с длинными целыми числами, использующимися в несимметричной криптографии.

Постановка задачи: пусть a, b – неотрицательные целые числа, десятичная запись которых может достигать 100 знаков. Написать программу, реализующую:

1. Арифметические операции над длинными целыми a, b.
   1) $a + b$ – сложение
   2) $a - b$ – вычитание (при $a < b$ считать $a - b = 0$)
   3) $a \times b$ – умножение
   4) $a \div b$ – деление нацело
   5) $a \bmod b$ – остаток от деления
   6) a^b – возведение в степень

   Реализовать операции в виде библиотеки функций (классов). Для сложения, вычитания, умножения и деления можно использовать школьные алгоритмы «в столбик». Возведение в степень рекомендуется делать алгоритмом быстрого возведения, сложность которого $O(\log_2 n)$. Например, $a^{26} = (a^{13})^2 = (aa^{12})^2 = (a(a^6)^2)^2 = (a((a^3)^2)^2)^2 = (a((aa^2)^2)^2)^2$.
2. Нахождение наибольшего общего делителя НОД(a,b) при помощи алгоритма Евклида.
3. Проверка, является ли длинное целое a простым. Если a составное, разложить его на простые множители. Перебор простых множителей можно закончить при достижении величины $[\sqrt{a}]$, которую можно рассчитать приближенно (здесь квадратными скобками обозначается взятие целой части числа).

Хранить длинные целые удобно в массиве переменной длины, размещая младшие разряды в начале массива. Следует учесть, что при умножении и возведении в степень длина результата будет сильно увеличиваться. Производительность программы можно значительно повысить, если представлять числа в системе счисления по основанию 2^{32} вместо десятичной. Тогда в качестве элементов массива можно использовать 32-битные целые числа. (Здесь имеет смысл выбирать показатель степени, равный разрядности процессора.)

Метод проверки: выполнение контрольных вычислений по пп.1–3, сравнение результатов с эталонными.

Курсовая работа.

Программа для курсовой работы строится на основе лабораторной работы №7. Программа должна выполнять многораундовое шифрование файлов с использованием 4 криптоалгоритмов (№3), контроль целостности по цифровой подписи (№5), использовать собственный генератор псевдослучайных последовательностей разрядности не менее 64 бит (№6), поддерживать два способа генерации ключей (№7). Предусмотреть обработку ошибок времени исполнения – не найден входной или ключевой файл, невозможно создать выходной или ключевой файл, неверная структура ключевого файла, недостаточно памяти или иных ресурсов и т.п.

Допускается реализация одного из двух видов интерфейса пользователя.

1. Графический интерфейс. Реализуется как графическое (оконное) приложение с кнопками, полями ввода и прочими элементами управления. Рассчитано на взаимодействие с пользователем в диалоговом режиме. Интерфейс должен быть простым и понятным даже для неподготовленных пользователей. Должна быть возможность указать имена обрабатываемых и ключевых файлов. Сообщения об ошибках выводятся в диалоговых окнах или иным способом. Примерами подобных приложений являются GUI-версии WinRAR или 7-Zip.

2. Консольный интерфейс. Реализуется как консольное приложение с передачей параметров через аргументы командной строки и выдачей сообщений в поток стандартного вывода (stdout). При нормальной работе программа возвращает код завершения 0. В случае ошибки программа завершается с ненулевым кодом и выводит на stdout (или stderr) сообщение об ошибке. Интерфейс программы должен допускать её применение в полностью автоматическом режиме – для вызова из других программ или сценариев. Приветствуется реализация фильтра, сочетающего ввод аргументов из командной строки, прием входных данных из stdin, вывод выходных данных в stdout и сообщений в stderr. Примерами подобных приложений являются консольный rar, tar или gzip.

После реализации и отладки программы требуется проверить качество шифрования. Выбираем один–два файла каждого типа:

1. Несжатый текст (*.txt, *.html)
2. Несжатый звук (*.wav, формат PCM)
3. Несжатые изображения (*.bmp, *.raw)
4. Документы MS Office (*.doc, *.xls)
5. Таблицы баз данных (*.dbf, *.mdb, MySQL, PostgreSQL)
6. Исполняемые файлы (*.exe, *.dll, форматы PE, ELF)
7. Сжатый звук (*.mp3, *.ogg)
8. Сжатые изображения (*.jpg, *.gif, *.png)
9. Сжатое видео (*.mpg, *.avi, кодеки DivX, XviD, H.264)
10. Архивы (*.zip, *.rar, *.7z, *.gz, *.bz2)

Эти файлы шифруются при помощи реализованной программы. Закрытые файлы тестируем программой криптографических тестов (лабораторная работа №4). Результаты тестов прикладываются к пояснительной записке.

Приложение №1.
Метод Гаусса приведения
матрицы к верхнетреугольному виду.

Рассмотрим метод Гаусса на примере матрицы 4×4 в общем виде.

За первый проход обнулим все элементы первого столбца матрицы, кроме элемента первой строки. Среди всех строк матрицы выбираем строку i_m с максимальным по модулю первым элементом: $a_{i_m 1} = \max\limits_{i\in[1;4]}|a_{i1}|$. Поменяем местами найденную строку и первую. Это позволит сократить вычислительную погрешность. Если все a_{i1} равны нулю, тогда матрица является вырожденной, и ее определитель равен нулю.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Почленно прибавим ко второй строке первую, умноженную на $\left(-\frac{a_{21}}{a_{11}}\right)$. Матрица примет вид

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix},$$

где $a'_{22} = a_{22} - \frac{a_{21}}{a_{11}}a_{12}$, $a'_{23} = a_{23} - \frac{a_{21}}{a_{11}}a_{13}$, $a'_{24} = a_{24} - \frac{a_{21}}{a_{11}}a_{14}$ – новые значения элементов второй строки матрицы. Почленно прибавим к третьей строке первую, умноженную на $\left(-\frac{a_{31}}{a_{11}}\right)$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix},$$

где $a'_{32} = a_{32} - \frac{a_{31}}{a_{11}}a_{12}$, $a'_{33} = a_{33} - \frac{a_{31}}{a_{11}}a_{13}$, $a'_{34} = a_{34} - \frac{a_{31}}{a_{11}}a_{14}$. Почленно прибавим к четвертой строке первую, умноженную на $\left(-\frac{a_{41}}{a_{11}}\right)$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{pmatrix},$$

где $a'_{42} = a_{42} - \frac{a_{41}}{a_{11}}a_{12}$, $a'_{43} = a_{43} - \frac{a_{41}}{a_{11}}a_{13}$, $a'_{44} = a_{44} - \frac{a_{41}}{a_{11}}a_{14}$. Теперь все элементы первого столбца ниже главной диагонали равны нулю. Первый проход на этом завершен.

Второй проход аналогичен первому. Теперь мы работаем с матрицей меньшего размера, и обнуляем элементы второго столбца, лежащие ниже главной диагонали. Выбираем из

оставшихся строк строку i'_m с максимальным по модулю вторым элементом $a'_{i'_m 2} = \max\limits_{i \in [2;4]} |a'_{i2}|$ и меняем ее со второй. Почленно прибавим к третьей строке вторую, умноженную на $\left(- \frac{a'_{32}}{a'_{22}} \right)$, а к четвертой – вторую, умноженную на $\left(- \frac{a'_{42}}{a'_{22}} \right)$. Матрица примет вид

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & a''_{43} & a''_{44} \end{pmatrix}.$$

В третьем проходе обнуляется последний элемент ниже главной диагонали. В случае $a''_{i''_m 3} = \max\limits_{i \in [3;4]} |a''_{i3}| \neq a''_{33}$, третья и четвертая строки меняются местами. Затем почленно прибавим к четвертой строке третью, умноженную на $\left(- \frac{a''_{43}}{a''_{33}} \right)$. Матрица принимает окончательный вид

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & 0 & a'''_{44} \end{pmatrix}.$$

Все элементы главной диагонали ненулевые, все элементы ниже главной диагонали равны нулю. Такая матрица является невырожденной, и ее определитель не равен нулю.

Приложение №2.
Умножение векторов и матриц.

Рассмотрим матрицу A размером $n \times n$ и вектор x длины n. Результатом умножения матрицы A на вектор x будет вектор y длиной n: $Ax = y$.

$$\begin{pmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \dots & a_{nj} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix},$$

где $y_i = \sum_{k=1}^{n} a_{ik} x_k \quad \forall i \in [1; n]$. Нельзя умножать матрицу на вектор, если ширина матрицы не равна длине вектора.

Рассмотрим две матрицы A и B размером $n \times n$. Результатом их умножения матрицы A на матрицу B будет матрица C размером $n \times n$: $A \times B = C$.

$$\begin{pmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \dots & a_{nj} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} b_{11} & \dots & b_{1j} & \dots & b_{1n} \\ \vdots & & \vdots & & \vdots \\ b_{i1} & \dots & b_{ij} & \dots & b_{in} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & \dots & b_{nj} & \dots & b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1j} & \dots & c_{1n} \\ \vdots & & \vdots & & \vdots \\ c_{i1} & \dots & c_{ij} & \dots & c_{in} \\ \vdots & & \vdots & & \vdots \\ c_{n1} & \dots & c_{nj} & \dots & c_{nn} \end{pmatrix},$$

где $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \forall i, j \in [1; n]$. Умножение матриц не коммутативно: $AB \neq BA$. Нельзя умножать матрицы, если ширина первой матрицы не равна высоте второй.

Список литературы.

11. Иванов М.А. Криптографические методы защиты информации в компьютерных системах и сетях.
12. Шнайер Б. Прикладная криптография.