ИНСТИТУТ ФИЗИКИ
КАЗАНСКОГО (ПРИВОЛЖСКОГО) ФЕДЕРАЛЬНОГО
УНИВЕРСИТЕТА

КАРПОВ А.В., ЛЮБИМОВ Д.В., СУЛИМОВ А.И.

ВВЕДЕНИЕ В КРИПТОГРАФИЮ

(учебно-методическое пособие
для выполнения лабораторных работ)

Казань – 2013

Печатается по решению Учебно-методической комиссии Института физики Казанского (Приволжского) Федерального Университета
Протокол № 3 от « 7 » июня 2013 г.

УДК 004.056.55, 003.26.09, 004.421.5

Карпов А.В., Любимов Д.В., Сулимов А.И. Введение в криптографию. Учебно-методическое пособие для выполнения лабораторных работ. Казань, 2013. 37 с.

В пособии изложены основы защиты информации криптографическими методами. Каждый раздел содержит необходимый теоретический материал и подробные указания для выполнения практических заданий на компьютере. Пособие предназначено для подготовки к выполнению лабораторных и практических работ, а также курсовых/дипломных проектов и магистерских диссертаций.

Учебно-методическое пособие предназначено для студентов, обучающихся по специальности 090900.62 – «Защита информации» по дисциплине «Криптографические методы защиты информации» и обучающихся по специальности 011800.68 – «Радиофизика» в магистратуре «Информационные процессы и системы» по дисциплине «Основы информационной безопасности».

Рецензент: Ишмуратов Р.А. – к.ф.-м.н., доцент кафедры Информатики и информационно-управляющих систем Казанского государственного энергетического университета.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящее пособие содержит описание четырёх лабораторных работ по основам криптографии. Выполнение каждой лабораторной работы предполагает разработку ряда компьютерных программ, реализующих изучаемые криптографические алгоритмы. По результатам выполнения каждой лабораторной работы составляется отчёт, который представляется в виде электронного документа в формате doc, docx или odt. При оформлении отчёта рекомендуется придерживаться приведённой ниже структуры.

Структура отчёта

1. Название и номер лабораторной работы.
2. Фамилия студента, номер учебной группы.
3. Цель работы.
4. Краткие теоретические сведения.
5. Описание хода лабораторной работы.
6. По возможности рекомендуется дополнять каждый пункт лабораторного задания снимками рабочего окна программы, иллюстрирующими выполнение задания.
7. Изложение результатов по каждому пункту лабораторного задания должно завершаться подведением основных выводов.
8. Текст разработанных программ приводится в последнем разделе отчёта.

Исторические шифры

Лабораторная работа №1

Программная реализация шифра замены

Шифр замены является одним из самых первых известных типов шифров [1,2]. Его работа основана на замене каждой буквы шифруемого сообщения какой-то другой фиксированной буквой. Перед началом шифрования выбирается *нормативный алфавит*, то есть набор символов, которые будут подвергаться шифрованию (замене). Обычно этот алфавит совпадает с алфавитом языка, на котором написан текст, однако он может включать в себя и другие символы, такие как цифры, знаки препинания, пробел и т.д. Символы, не входящие в нормативный алфавит, не подвергаются замене (остаются в тексте без изменения). Замена производится на основе заранее выбранной таблицы, которая играет роль ключа шифрования: каждый символ из левой части таблицы заменяется соответствующим символом из правой части таблицы. В качестве примера рассмотрим следующую таблицу.

А	Е
Б	Р
В	Й
Г	О
Д	М
Е	Т
…	…
Я	Б
пробел	П

При её использовании фраза «где я» будет зашифрована строкой «омтпб». Таблица замены работает в обе стороны, то есть отображение является взаимно однозначным. Это позволяет использовать ту же таблицу для расшифровки текста. В этом случае замена выполняется в обратном порядке (справа налево).

Замечание: заглавные и строчные буквы заменяются по одной и той же таблице.

Недостатком шифра замены является его слабая стойкость к криптоанализу (подбору ключа к зашифрованному сообщению). Общий подход к взлому шифра замены заключается в статистическом анализе зашифрованного текста и сравнении полученной статистики с характеристиками языка, на котором написан исходный текст. Известно, что различные буквы в тексте на естественном языке встречаются с разной частотой. Так, в русском языке наиболее распространенными являются буквы О, Е, А, И. На основе анализа большого числа текстов на русском языке были найдены частоты встречаемости всех букв. Ниже приведена таблица, в которой буквы расположены в порядке убывания частоты их появления в русском языке (буква «Ё» считается совпадающей с буквой «Е»).

Символ	Частота	Символ	Частота	Символ	Частота
Пробел	0,175	К	0,028	Ч	0,012
О	0,089	М	0,026	Й	0,010
Е	0,072	Д	0,025	Х	0,009
А	0,062	П	0,023	Ж	0,007
И	0,062	У	0,021	Ю	0,006
Т	0,053	Я	0,018	Ш	0,006
Н	0,053	Ы	0,016	Ц	0,004
С	0,045	З	0,016	Щ	0,003
Р	0,040	Ь, Ъ	0,014	Э	0,003
В	0,038	Б	0,014	Ф	0,002
Л	0,035	Г	0,013		

Рассмотрим, как производится статистический частотный криптоанализ (взлом) шифра замены. На ***первом шаге*** находятся частоты

встречаемости всех символов зашифрованного сообщения. Частота символа x находится по формуле:

$$q(x) = \frac{N(x)}{N},$$

где $N(x)$ – количество символов x в зашифрованном тексте, а N – общее количество символов в сообщении.

На ***втором шаге*** символы шифротекста записываются в таблицу в порядке убывания их частоты. Это позволяет сопоставить частотную таблицу шифротекста с частотной таблицей русского языка.

На ***третьем шаге*** производится последовательная замена (расшифровка) символов шифротекста, начиная с верхней части таблицы. Так, символ с самой большой частотой скорее всего соответствует пробелу, а следующий по частоте символ – букве О. Дальнейшая расшифровка текста на основе частот отдельных символов становится затруднительной, поскольку частоты соседних символов близки друг к другу.

Для получения дополнительной статистической информации анализируются частоты биграмм. *Биграммой* называют последовательность из двух идущих подряд букв. Наиболее распространенные в тексте биграммы сравниваются с самыми распространенными биграммами русского языка. К ним относятся биграммы: СТ, ТО, ОВ, НА, НЕ, ЕН, НО, ВО, АЛ, НИ, ПО, РА. Видно, что большинство распространенных биграмм состоит из одной гласной и одной согласной буквы.

Кроме биграмм, также можно найти самые распространенные *триграммы* (последовательности из трех букв). В русском языке распространены триграммы: СТО, ЕНИ, ОГО, ОСТ, ЧТО.

Также полезную информацию могут дать биграммы, состоящие из одинаковых букв (наиболее часто встречаются НН, ЕЕ, СС, ИИ), а также слова, состоящие из одной буквы (И, В, С, К).

Подбор ключа (таблицы замены) осуществляется по одной букве с постоянным контролем полученного текста. В случае обнаружения ошибочной замены (например, получено несуществующее слово) необходимо скорректировать одну или несколько замен.

Задания по лабораторной работе №1

Задание 1. Написать программу, реализующую шифр замены. Требования к программе:

- В начале работы программы пользователь выбирает тип операции: шифрование или расшифрование.
- Программа позволяет зашифровывать и расшифровывать текст на русском языке, записанный в текстовом файле в стандартной кодировке ОС Windows.
- Зашифрованный текст сохраняется в отдельном текстовом файле.
- Таблица замены (ключ шифрования) генерируется случайным образом и сохраняется в файле key.txt.
- Расшифрование файла производится на основе таблицы, которая считывается из файла key.txt.
- Расшифрованный текст сохраняется в отдельном файле.

Задание 2. Расшифровать текст, выданный преподавателем, с помощью частотного криптоанализа (т.е. при отсутствии ключа шифрования). Текст получен путём замены букв и пробелов по некоторой (заранее неизвестной) таблице. Остальные символы (знаки препинания, цифры) остаются неизменными.

В результате работы должна быть получена частотная таблица выданного шифротекста, а на её основе подобрана таблица замены. Частотная таблица находится с помощью вспомогательной программы.

Контрольные вопросы

1. Как осуществляются шифрование и расшифрование текста с помощью шифра замены?
2. Что такое ключ шифрования?
3. Легко ли подобрать ключ шифра замены методом прямого перебора? Сколько существует всевозможных таблиц замены?
4. В чем состоит уязвимость шифра замены?
5. Что такое частотная таблица и как она получается?
6. Как осуществляется частотный криптоанализ шифра замены?

Поточные шифры

Лабораторная работа №2

Генерирование и тестирование псевдослучайных шифрующих последовательностей

Поточный шифр – это шифр, в котором каждый символ открытого текста шифруется независимо от других символов с помощью потока ключей, который называется *ключевой последовательностью* (а также *гамма-последовательностью,* или просто *гаммой*). Ключевая последовательность произвольной длины получается из короткого ключа фиксированной длины с помощью некоторого алгоритма.

Обычно шифрование сводится к *побитовому* сложению по модулю 2 открытого текста и ключевой последовательности:

$$шифротекст = текст \oplus ключ. \quad (1)$$

Символ $\oplus$ обозначает операцию «сложение по модулю 2» битов сообщения и ключевой последовательности. Также эта операция называется «исключающее ИЛИ», а сама процедура шифрования называется *гаммированием*.

Расшифрование основано на одном из свойств операции $\oplus$:

$$(m \oplus k) \oplus k = m. \quad (2)$$

Отсюда следует, что для расшифрования нужно выполнить сложение по модулю 2 зашифрованного сообщения с той же самой ключевой последовательностью, что использовалась при шифровании.

Регистр сдвига с линейной обратной связью

Распространенный способ генерации ключевой последовательности основан на использовании регистра сдвига с обратной связью [1]. Он состоит из двух частей: регистра сдвига и функции обратной

связи. *Регистр сдвига* – это последовательность однобитовых ячеек памяти фиксированной длины, значения которых могут сдвигаться в одном направлении. Количество однобитовых ячеек (разрядов) называется *длиной* регистра, или *разрядностью*.

Функция обратной связи является некоторой булевой функцией от содержимого всех двоичных разрядов регистра и может принимать одно из двух значений: 0 или 1. Процесс генерации потока битов показан на Рис. 1.

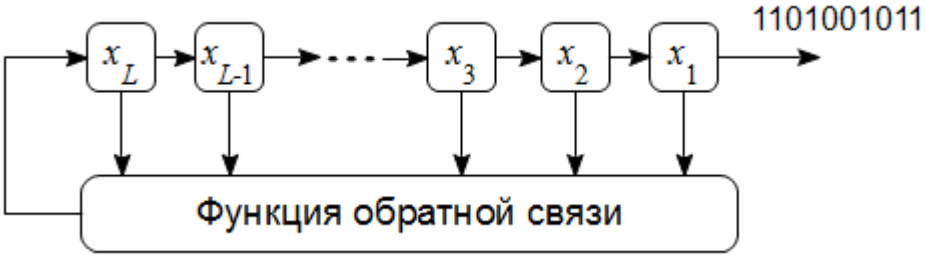

Рис. 1. Структура регистра сдвига с обратной связью.

В этой схеме разряды регистра обозначаются как $x_1, x_2, x_3, \cdots, x_L$, где L – длина регистра. Работа регистра разбивается на такты. На каждом такте происходит вычисление функции обратной связи $y = F(x_1, x_2, \cdots, x_L)$, после чего содержимое регистра сдвигается вправо на один разряд, т.е. $x_{L-1} = x_L$, $x_{L-2} = x_{L-1}, \ldots, x_1 = x_2$. После этого значение y записывается в крайний левый (старший) разряд x_L, а значение x_1 (до сдвига) «выталкивается» из регистра и становится частью выходной двоичной последовательности.

Простейшим видом функции обратной связи является линейная функция, являющаяся суммой по модулю два некоторых заранее выбранных фиксированных разрядов регистра. Такой регистр называется *регистром сдвига с линейной обратной связью* (Linear Feedback Shift Register, сокращенно LFSR).

В общем случае линейная функция обратной связи задается формулой $F(x) = c_L x_L \oplus c_{L-1} x_{L-1} \oplus \cdots \oplus c_1 x_1$. Здесь $c_k = 1$, если k-й разряд используется в функции обратной связи, иначе $c_k = 0$.

В качестве примера рассмотрим LFSR длины $L = 4$ с функцией обратной связи $F(x) = x_4 \oplus x_1$, где $\oplus$ – сложение по модулю 2 (см. Рис. 2).

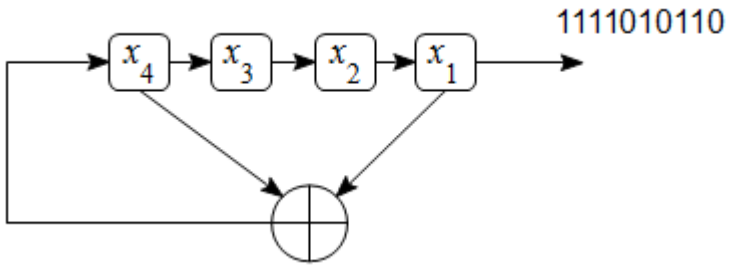

Рис. 2. Пример регистра сдвига с линейной обратной связью.

На каждом такте производится сложение по модулю 2 первого и четвертого разрядов регистра. Если начальным состоянием регистра является 1111, то на каждом такте он будет принимать значения: 1111, 0111, 1011, 0101, 1010, 1101, 0110, 0011, 1001, 0100, 0010, 0001, 1000, 1100, 1110, 1111, …

Выходная последовательность формируется из младшего (крайнего правого) разряда регистра. Она будет выглядеть следующим образом: 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 1…

Видно, что генерируемая битовая последовательность целиком определяется начальным состоянием регистра и функцией обратной связи. Поскольку число всевозможных состояний регистра конечно (оно равно 2^L), то, рано или поздно, ключевая последовательность начнёт повторяться. Максимальная длина неповторяющейся части ключевой последовательности называется ее *периодом*. Период зависит от длины регистра и от функции обратной связи. Максимально возможный период равен 2^L-1. Выходная последовательность LFSR, обладающая таким периодом, называется *M-последовательностью*.

Чтобы выяснить условия, при которых LFSR будет обладать максимальным периодом, функции обратной связи ставят в соответствие полином $P(x) = c_L x^L + c_{L-1} x^{L-1} + \cdots + c_1 x + 1$. Тогда регистру, приведенному на Рис. 2, соответствует полином $P(x) = x^4 + x^1 + 1$. Теоретический анализ показывает, что LFSR будет обладать максимальным периодом тогда и только тогда, когда полином $P(x)$ является *примитивным* (неприводимым по модулю 2). Ни-

же приведены некоторые примитивные полиномы, рекомендованные к применению на практике. Количество степенных слагаемых в полиноме соответствует числу отводов регистра. В целях увеличения быстродействия процедуры генерирования ключевой последовательности желательно выбирать полиномы с наименьшим числом слагаемых: $P(x) = x^m + x^l + 1$.

$$x^{31}+x^3+1 \quad x^{31}+x^6+1 \quad x^{31}+x^7+1 \quad x^{33}+x^{13}+1$$
$$x^{71}+x^7+1 \quad x^{93}+x^2+1 \quad x^{137}+x^{21}+1 \quad x^{35}+x^2+1$$
$$x^{145}+x^{52}+1 \quad x^{161}+x^{18}+1 \quad x^{521}+x^{32}+1 \quad x^{47}+x^5+1$$
$$x^{55}+x^{24}+1 \quad x^{58}+x^{19}+1 \quad x^{57}+x^7+1 \quad x^{52}+x^{49}+1$$

Для увеличения быстродействия состояние регистра выгодно хранить в виде целого L-разрядного числа, отдельные биты которого соответствуют двоичным разрядам регистра. Тогда для доступа к отдельным битам используются побитовые операции (сдвиг, маскирование и т.д.).

Статистические свойства М-последовательности

Двоичная последовательность, выдаваемая LFSR, должна удовлетворять определённым требованиям. С криптографической точки зрения важным требованием является невозможность предсказать следующий бит последовательности при известных предыдущих битах. К сожалению, для LFSR это условие не выполняется: если злоумышленник знает L идущих подряд битов последовательности, а также образующий полином, он может точно предсказать всю остальную последовательность. Даже если образующий полином неизвестен, для его определения достаточно знать $2L$ битов последовательности. В связи с этим на практике используют комбинации из нескольких LFSR разной длины.

Хорошая ключевая последовательность должна быть неотличима от *случайной равномерно распределённой* двоичной последовательности. Поскольку *M*-последовательность не является случайной, будем называть её *псевдослучайной* (то есть похожей на случайную). В статистике был разработан ряд критериев случайности последовательности, которые на практике реализуются в виде статистических тестов. Рассмотрим основные эмпирические тесты [3].

Сериальный тест

В сериальном тесте проверяется равномерность распределения *серий* из нескольких последовательных элементов выборки. Например, рассмотрим серии из двух идущих подряд битов. Всего будет 4 различных комбинаций: 00, 01, 10, 11. Вероятность появления каждой комбинации одинакова и равна 0,25. В случае серии длиной *k* бит вероятность будет равна $p = \frac{1}{2^k}$. При проведении сериального теста последовательность из *M* битов разбивается на $N = \frac{M}{k}$ непересекающихся серий. Для каждой двоичной комбинации определяется частота $N_j^Э$, то есть количество серий, совпадающих с *j*-й комбинацией ($j = 1, \ldots, 2^k$). Теоретическая частота каждой комбинации будет равна $N_j^T = \frac{N}{2^k}$, поскольку все комбинации равновероятны.

Если тестируемая двоичная последовательность удовлетворяет сериальному тесту, то значения $N_j^Э$ будут близки к N_j^T. В качестве статистического критерия близости $N_j^Э$ к N_j^T используется критерий χ^2 Пирсона с $2^k - 1$ степенями свободы:

$$\chi^2 = \sum_{j=1}^{2^k} \frac{(N_j^Э - N_j^T)^2}{N_j^T}. \qquad (3)$$

Чем меньше значение χ^2, тем равномерней распределение битов в последовательности. Если χ^2 превысит некоторый порог $\chi^2_{кр}$, который называется *критическим значением*, то говорят, что последовательность не прошла сериальный тест (недостаточно равномерна). Для нахождения $\chi^2_{кр}$ нужно задать *уровень значимости*, то есть такую вероятность α, что

$$P(\chi^2 > \chi^2_{кр}) = \alpha. \qquad (4)$$

Обычно α выбирают равным 0,1 или 0,05. Для заданного уровня значимости находится верхняя граница χ^2 с помощью Таблицы 1. Однако, наряду с верхней границей, необходимо задать и нижнюю границу. Это связано с тем, что слишком хорошее согласие с критерием χ^2 (если значение χ^2 очень мало) также характеризует последовательность как недостаточно случайную. Таким образом, нужно задать два уровня значимости (высокий и низкий), например 0,1 и 0,9. Затем для каждого уровня значимости вычисляется критическое значение параметра χ^2 и проверяется выполнение неравенства

$$\chi^2_{кр}(\alpha = 0{,}9) < \chi^2 < \chi^2_{кр}(\alpha = 0{,}1)\,. \qquad (5)$$

При выполнении этого неравенства считается, что последовательность удовлетворяет критерию хи-квадрат, а значит проходит сериальный тест.

Таблица 1

Критические значения параметра χ^2 для серий 2, 3 и 4 бит

степени свободы	Уровень значимости α			
	0,95	**0,9**	**0,1**	**0,05**
3	0,352	0,584	6,251	7,815
7	2,167	2,833	12,017	14,067
15	7,261	8,547	22,307	24,996

Корреляционный тест

В данном тесте исследуются корреляционные свойства последовательности с помощью коэффициента автокорреляции. Коэффициент корреляции двух последовательностей $\{x_i\}$ и $\{y_i\}$ длины N находится по формуле

$$R = \frac{N\sum_{i=1}^{N} x_i y_i - S_x S_y}{\sqrt{(N\sum_{i=1}^{N} x_i^2 - S_x^2)(N\sum_{i=1}^{N} y_i^2 - S_y^2)}}\,, \qquad (6)$$

где $S_x = \sum_{i=1}^{N} x_i$, $S_y = \sum_{i=1}^{N} y_i$ (7)

Чтобы найти автокорреляцию, нужно положить $y_i = x_{i+k}$, $k = 1,2,\ldots$ Большой по модулю коэффициент R_k говорит о сильной статистической связи между i-м и $(i+k)$-м членами последовательности, что нежелательно. Для численной оценки величины R_k используют неравенство (для α=0,05)

$$|R| \le \frac{1}{N-1} + \frac{2}{N-2}\sqrt{\frac{N(N-3)}{N+1}} \quad (8)$$

Задания по лабораторной работе №2

Задание 1. Написать программу, реализующую генератор M-последовательности на основе полинома обратной связи, выданного преподавателем. Начальные значения битов регистра сдвига задаются случайным образом (с помощью стандартного генератора случайных чисел). В программе должна быть возможность сохранять начальное состояние регистра сдвига в файле key.txt, а также возможность загружать начальное состояние регистра из этого же файла. Предусмотреть возможность вывода первых n двоичных элементов M-последовательности на экран (n задаётся пользователем).

Примечание. Предварительно проверить правильность работы алгоритма для полинома $P(x) = x^4 + x^1 + 1$ (см. Рис. 2).

Задание 2. Протестировать M-последовательность длиной N бит ($N \le 10000$, задаётся преподавателем) с помощью сериального теста. Использовать серии длиной 2, 3 или 4 бита (по указанию преподавателя). Найти χ^2 по формуле (3) и проверить выполнение неравенства (5) для заданного уровня значимости α.

При тестировании (Задание 2, 3) необходимо учесть тот факт, что первые L битов M-последовательности совпадают с начальным состоянием регистра сдвига и не зависят от функции обратной связи. Если начальные разряды реги-

стра недостаточно «случайны», результаты тестов могут быть искажены. Чтобы этого избежать, первые $2L$ тактов регистр должен работать в «холостом режиме», то есть первые $2L$ битов не участвуют в тестировании.

Задание 3. Протестировать M-последовательность с помощью корреляционного теста. Найти значение R_k по формуле (6) для $k = 1, 2, 8, 9$. Проверить выполнение неравенства (8).

Задание 4. Реализовать программу шифрования/расшифрования двоичного файла с помощью M-последовательности. В качестве ключа использовать начальное значение регистра сдвига, которое хранится в файле key.txt. Результат шифрования сохранить в файле encoded.txt. Расшифровать файл encoded.txt с помощью того же ключа и убедиться, что расшифрованный и исходный файлы идентичны. Шифрование и расшифрование осуществляется по формуле (1) путём сложения по модулю два (операция «исключающее ИЛИ») двоичного представления шифруемого файла с M-последовательностью.

Задание 5. Протестировать исходный и зашифрованный файлы с помощью статистических тестов (Задание 2, 3). Объяснить результаты тестов.

Контрольные вопросы

1. Что такое поточный шифр?
2. Как осуществляется поточное шифрование и расшифрование?
3. Что такое регистр сдвига с линейной обратной связью?
4. Что такое M-последовательность? Как её получить?
5. Какими свойствами должна обладать шифрующая ключевая последовательность?
6. Как проводится сериальный тест?
7. Для чего и как используется критерий χ^2?
8. Что такое коэффициент автокорреляции? Какие он может принимать значения?

Криптосистемы с открытым ключом

Лабораторная работа №3

Асимметричное шифрование на основе алгоритма RSA

Асимметричную схему шифрования иначе называют шифрованием с открытым ключом. В данной схеме для шифрования и расшифрования данных используются два различных ключа. Первый ключ, называемый *открытым ключом*, используется только для шифрования данных. Второй ключ, называемый *закрытым ключом*, используется только для расшифрования данных, зашифрованных соответствующим открытым ключом. Поскольку открытый и закрытый ключ неразрывно связаны друг с другом, они должны генерироваться одновременно.

Открытый ключ может свободно передаваться по незащищённым каналам связи, поскольку с его помощью нельзя расшифровать секретные данные. Напротив, закрытый ключ хранится получателем сообщения в секрете. Поскольку открытый ключ может быть перехвачен, он может быть использован злоумышленником для отправки зашифрованных сообщений. В связи с этим возникает задача идентификации отправителя сообщения.

Исторически первым и наиболее распространённым алгоритмом асимметричного шифрования является алгоритм RSA. Его надёжность основана на вычислительной сложности задачи *факторизации*, то есть разложения целых чисел на простые множители. В то же время, прямая задача (нахождение числа по его простым множителям) является тривиальной. Рассмотрим подробнее процедуру генерации пары ключей в алгоритме RSA и процедуру шифрования/расшифрования данных [1,2].

Процедура генерации открытого и закрытого ключей RSA

1. Выбираются два случайных *простых* числа p и q заданной битовой длины $L/2$, где L — длина ключа.
2. Вычисляется их произведение $n=p\cdot q$, которое называется *модулем шифрования*. Битовая длина n будет равна L.

3. Вычисляется функция Эйлера $\varphi(n) = (p-1)(q-1)$. Её значение равно количеству целых чисел a_k $(1 \le a_k < n)$, взаимно простых с n.
4. Выбирается целое число $e < n$, взаимно простое с $\varphi(n)$, то есть такое, что НОД($\varphi(n)$, e) = 1 (НОД - *наибольший общий делитель*). Число e называется *открытой экспонентой*. Обычно в качестве открытой экспоненты выбирают простые числа, содержащие малое число единиц в двоичной записи (например, *числа Ферма*: 3, 17, 257, 65537). Для вычисления наибольшего общего делителя применяется алгоритм Евклида.
5. **Пару чисел (e, n) используют в качестве открытого ключа.**
6. Для получения закрытого ключа вычисляется число d, удовлетворяющее условию $(e \cdot d) \bmod \varphi(n) = 1$. Число d называется *закрытой экспонентой*. Для нахождения d используется расширенный алгоритм Евклида. Запись $x \bmod n$ обозначает остаток от деления x на n (по-другому эту операцию называют делением по модулю n).
7. **Пару чисел (d, n) используют в качестве закрытого ключа.**

Шифрование и расшифрование данных по алгоритму RSA

Для шифрования сообщения M оно предварительно разбивается на блоки. Обычно длину блоков выбирают примерно равной $L/4$. Каждый i-й блок M_i шифруется по формуле

$$C_i = M_i^e \bmod n.$$

Обратная операция дешифрации блока C_i осуществляется по формуле

$$M_i = C_i^d \bmod n.$$

В этих формулах используется операция возведения в степень по модулю n. При возведении числа x в k-ю степень число умножается само на себя k-1 раз. После каждого умножения производится деление по модулю n. Для больших степеней простое перемножение оказывается очень затратным, поэтому на практике используют алгоритм быстрого возведения в степень.

Алгоритм быстрого возведения в степень

Идею быстрого возведения в степень можно пояснить на примере: $x^8 = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x = ((x^2)^2)^2$. Таким образом, вместо семи умножений достаточно только трёх. Приведенный ниже алгоритм особенно эффективен, если число единиц в двоичной записи степени минимально (таким свойством обладают числа Ферма).

```
входные данные: x, d, n
цель алгоритма: найти x^d (mod n)
начало алгоритма
y = 1;
цикл пока d > 0
    если d – нечётное то
          y = (y*x) mod n;
    конец если
    d = d div 2; // целочисленное деление на 2
    x = (x*x) mod n;
конец цикла
ответ: y
конец алгоритма
```

Алгоритм Евклида

Алгоритм Евклида позволяет найти наибольший общий делитель двух целых чисел a и b: c = НОД(a, b). В основе алгоритма Евклида лежит равенство НОД (a, b) = НОД (b, a mod b) . Замены (a, b) → (b, a mod b) производятся до тех пор, пока одно из чисел не станет равным нулю.

```
входные данные: целые числа a, b
выходные данные: НОД(a, b)
начало алгоритма
цикл пока b ≠ 0
    t = a mod b; // остаток от деления a на b
    a = b;
    b = t;
конец цикла
ответ: НОД = a
конец алгоритма
```

Расширенный алгоритм Евклида

Расширенный алгоритм Евклида позволяет найти разложение наибольшего общего делителя НОД(a, b) в виде линейной комбинации самих чисел a и b, то есть решить целочисленное уравнение НОД(a, b) = $as + bt$.

Результатом работы алгоритма является НОД(a, b), а также целочисленные коэффициенты s и t (которые могут быть отрицательными). Расширенный алгоритм Евклида позволяет найти решение уравнения $ax \bmod N = 1$, которое имеет единственное решение, если НОД(a, N) = 1. Результатом работы алгоритма будет разложение $1 = as + Nt$, которое иначе можно записать как $as \bmod N = 1$, то есть s является решением уравнения $ax \bmod N = 1$. Далее приводится псевдокод этого алгоритма.

```
Расширенный алгоритм Евклида
входные данные: m, n
выходные данные: d, s, t
цель алгоритма: найти d=НОД(m, n) и найти s, t такие,
что d = s*m + t*n
начало алгоритма
a = m; b = n;
u1 = 1; v1 = 0;
u2 = 0; v2 = 1;
цикл пока b ≠ 0
    q = a div b; // целочисленное деление a на b
    r = a mod b; // остаток от деления a на b
    a = b; b = r;
    r = u2;
    u2 = u1 – q*u2;
    u1 = r;
    r = v2;
    v2 = v1 – q*v2;
    v1 = r;
конец цикла
ответ: d = a; s = u1; t = v1;
конец алгоритма
```

Таким образом, чтобы найти d, удовлетворяющее уравнению $(e \cdot d) \bmod \varphi(n) = 1$, нужно выполнить расширенный алгоритм Евклида с входными дан-

ными $m = e$, $n = \varphi(n)$. Переменная s будет содержать искомое значение d. Если оно окажется отрицательным, к нему нужно прибавить $\varphi(n)$.

Генерирование простых чисел заданной длины

В процессе создания ключей RSA необходимо генерировать случайные *простые* числа заданной длины. Для этого сначала генерируется случайное число заданной длины, которое затем проверяется на простоту. Если сгенерированное число не является простым, процедура повторяется (пока не будет получено простое число).

Чтобы сгенерировать случайное число длиной L бит, оно (число) заполняется случайными битами, причём старший (L-й) бит должен равняться 1.

Чтобы узнать, является ли число простым, используют специальные вероятностные тесты, поскольку точная проверка занимает слишком много времени (при достаточно больших L). Простейшим вероятностным тестом является тест, основанный на *теореме Ферма* (*тест Ферма*).

Согласно теореме Ферма, если число T – простое, то для любого $a < T$ выполняется условие $a^{T-1} \bmod T = 1$. Чем больше различных чисел a удовлетворяют этому условию, тем выше вероятность того, что T – простое число. Если условие теоремы выполняется для k различных чисел $\{a_1, a_2, \ldots, a_k\}$, то число T является простым с вероятностью $\frac{2^k - 1}{2^k}$. Таким образом, проверка числа T на простоту сводится к получению k различных (случайных) чисел, не превосходящих T, для каждого из которых проверяется условие теоремы Ферма. Чем больше k, тем меньше вероятность ошибки. Так, для $k = 10$ вероятность того, что непростое число будет ошибочно признано простым, равна 0,1%.

Недостатком теста Ферма является то, что существуют числа-исключения, которые удовлетворяют теореме Ферма, но при этом не являются простыми. Их называют числами Кармайкла. Такие числа встречаются довольно редко и в учебных целях ими можно пренебречь.

На практике вместо теста Ферма используются другие тесты, лишённые этого недостатка (например, тест Миллера-Рабина).

Реализация арифметических операций

Длина ключей в алгоритме RSA может составлять сотни и тысячи бит. На данный момент безопасным считается ключ длиной не мене 1024 бит. Поэтому при реализации алгоритма RSA возникает необходимость в выполнении арифметических операций над очень большими целыми числами, заведомо превышающими длину машинного слова. Однако большинство современных процессоров позволяет непосредственно работать с числами длиной не более 64 бит. Чтобы обойти это ограничение, используются специальные алгоритмы, относящиеся к арифметике произвольной точности.

Оптимальная реализация алгоритмов произвольной точности является непростой задачей, поэтому на практике рекомендуется использовать готовые библиотеки. Для языка C++ могут быть рекомендованы библиотеки MPIR и BigInteger. Рассмотрим подробнее вторую из них.

Библиотека BigInteger

Данная библиотека предоставляет удобные средства работы с *целыми* числами произвольной длины на языке C++. Для представления таких чисел в библиотеке используются два специальных класса переменных: BigInteger (длинное число со знаком) и BigUnsigned (длинное число без знака). Для этих классов переопределены основные арифметические и побитовые операции, операция вывода в поток (ostream), что позволяет работать с длинными числами так же, как с переменными стандартного типа int. Кроме того, библиотека содержит реализацию некоторых вспомогательных целочисленных алгоритмов, которые описаны в заголовочном файле BigIntegerAlgorithms.hh.

Библиотека BigInteger состоит из 12-ти файлов. Для использования библиотеки необходимо добавить эти файлы к проекту и подключить заголовочный файл BigIntegerLibrary.h в основной программе. Ниже приводится фраг-

мент программного кода, иллюстрирующий пример использования библиотеки BigInteger.

```
#include <iostream>
#include <string>
#include "BigIntegerLibrary.h"

int main()
{
    BigUnsigned x, y, z; // три целых числа произвольной длины
    std::string s;
    std::cin >> s; // ввод числа в виде строки
    x = stringToBigUnsigned(s); // преобразование строки в длинное число
    y = (x+1)*(x-2)/x; // доступны 4 арифметических операции
    z = y % 2; // остаток от деления y на 2
    int i = z.toInt(); // преобразование длинного целого в тип int
    std::cout << "z = " << z << "\n"; // вывод длинного числа на экран
    return 0;
}
```

Разложение чисел на множители методом ρ-эвристики Полларда

Надёжность алгоритма RSA основана на сложности задачи факторизации больших целых чисел. Простые алгоритмы факторизации оказываются неэффективными даже для очень коротких ключей. Однако популярность алгоритма RSA стимулировала поиск новых эффективных методов факторизации. Одним из простых и эффективных методов факторизации является метод ρ-эвристики Полларда [4]. Дадим его краткое описание.

Для факторизации числа n используется последовательность псевдослучайных чисел $\{x_i\}$, не превосходящих n. Получить такую последовательность можно, например, с помощью рекурсивной формулы: $x_{i+1} = (x_i^2 + 1) \bmod n$ (x_0 выбирается произвольно). Для каждой пары (x_i, x_j) проверяется условие: НОД(n, $|x_i - x_j|$) > 1. Если оно выполняется, то найденный НОД будет равен одному из делителей числа n и работа алгоритма завершается. Для ускорения ра-

боты алгоритма рассматриваются не все пары (x_i, x_j), а только те, для которых j является степенью 2 (то есть 2, 4, 8, 16, …).

В заключение стоит отметить, что сложность факторизации определяется наименьшим из двух сомножителем. Например, если p имеет длину 32 бита, а q 128 бит, то длительность факторизации будет сопоставима со случаем, когда длина q также равна 32 битам. Поэтому на практике рекомендуется выбирать числа p и q имеющими одинаковую битовую длину.

Задания по лабораторной работе №3

Задание 1. Программно реализовать процедуру генерации открытого и закрытого ключей заданной длины L (128, 256, 512). В качестве открытой экспоненты использовать одно из чисел Ферма (17, 257, 65537). Сформированные ключи сохранить в файлы: открытый – в файл public.txt, а закрытый – в файл private.txt.

Задание 2. Программно реализовать процедуры шифрования и дешифрования согласно алгоритму RSA. Использовать открытый и закрытый ключи, сформированные в Задании 1.

На вход программы подается текстовая строка S длиной не более $L/8$ символов. Каждому символу строки сопоставляется 8-битный двоичный код согласно таблице кодировки ASCII. Биты всех символов последовательно объединяются в одно большое двоичное число M, которое подвергается шифрованию и расшифрованию. Например, исходная строка S = 'abc' преобразуется в 24-битное число M = 01100001’01100010’01100011. В результате шифрования ключом public.txt образуется двоичное число C, которое записывается в файл encrypted.txt.

На втором этапе полученное ранее число C считывается из файла encrypted.txt и расшифровывается с помощью закрытого ключа private.txt. Полученное число M' преобразуется в символьную строку S', которая записывается в выходной файл decrypted.txt. При правильной реализации процедур шиф-

рования/расшифрования исходная S и расшифрованная S' строки должны совпасть.

Задание 3. Проверить работоспособность алгоритма RSA в случае, когда одно из чисел p и q не является простым.

Задание 4. Написать программу, реализующую атаку на алгоритм RSA (вычисление закрытого ключа по известному открытому ключу) с использованием ρ-эвристики Полларда. Результатом работы программы должно быть разложение заданного числа n на два простых множителя p и q.

Провести атаку для различных длин ключа. Убедиться, что с ростом битовой длины ключа (т. е. числа $n = p \cdot q$) растет время его факторизации. Построить график зависимости этого времени от длины ключа (пока время работы программы не превысит нескольких минут). Для этого последовательно увеличивать длину p и q, начиная с 30 бит с шагом в 2 бита. Когда время расчетов превысит 10 секунд, уменьшить шаг до 1 бита.

Задание 5. Выяснить, как влияет различие битовой длины чисел p и q на сложность факторизации. Для этого берется максимальная длина ключа, найденная в Задании 4, которую обозначим буквой L. Затем генерируются простые числа p и q, имеющие длину rL и $(1–r)L$ соответственно ($r < 1$). Построить график зависимости времени факторизации числа $n = p \cdot q$ от r для $r = 0{,}25 \ldots 0{,}5$ с шагом 0,05.

Контрольные вопросы

1. Для чего используются два ключа? В чем их различие?
2. Преимущества асимметричных схем шифрования.
3. Недостатки асимметричных схем шифрования.
4. На чем основана надежность криптосистемы *RSA*?
5. Как производится быстрое возведение в степень?
6. Что такое числа Кармайкла?

Криптографическая хэш-функция

Лабораторная работа №4

Реализация и исследование свойств алгоритмов хэширования

Функция хэширования – это функция, которая принимает на вход строку битов (или байтов) произвольной длины и выдаёт результат фиксированной длины. Результат работы хэш-функции называется *хэш-кодом* (или просто *хэшем*) сообщения. Хэш-функции находят широкое применение в теории кодирования, передачи информации, в разработке программного обеспечения. Примером простейших хэш-функций являются контрольные суммы и CRC-коды. Основное их назначение – обнаружение случайных повреждений данных. В криптографических приложениях к хэш-функциям предъявляются более строгие требования [1,2]. Функции, отвечающие этим требованиям, называются *криптографическими хэш-функциями*. Наиболее известными алгоритмами хэширования являются алгоритмы семейств MD и SHA. Ниже приведены основные области применения криптографических хэш-функций.

- Проверка целостности сообщения.
- Электронная цифровая подпись.
- Хранение и проверка паролей.

Для дальнейшего рассмотрения хэш-функций введем следующие понятия. Пусть M – сообщение, H – функция хэширования, $h = H(M)$ – хэш-код.

Прообразом h называется любое сообщение M', чей хэш-код равен h, то есть $H(M') = h$, где $M' = H^{-1}(h)$.

Сообщение M', имеющее такой же хэш-код, что и заданное сообщение M (т.е. $H(M') = H(M)$), называется *прообразом второго рода*.

Два сообщения M' и M'', имеющие одинаковый хэш-код (то есть $H(M') = H(M'')$), называют *коллизией*.

Отличие прообраза второго рода от коллизии заключается в том, что сообщение M известно заранее, в то время как сообщения M' и M'', образующие коллизию, могут быть произвольными.

Свойства криптографических хэш-функций

1. Эффективность: зная M, легко вычислить h, то есть алгоритм вычисления хэш-кода является эффективным (быстрым).

2. Односторонность: вычислительно сложно найти прообраз данного хэш-кода h. Под вычислительной сложностью понимается отсутствие более эффективного алгоритма, чем метод прямого перебора (т.е. перебор различных сообщений, пока не встретится сообщение с данным хэш-кодом). Вычислительная сложность прямого перебора зависит от длины хэш-кода: если хэш-код имеет длину n бит, то для нахождения сообщения с таким хэш-кодом потребуется перебрать в среднем 2^n различных сообщений.

3. Защищенность от коллизий: нахождение коллизии является вычислительно сложной задачей. Для нахождения коллизии методом прямого перебора потребуется перебрать в среднем $2^{n/2}$ сообщений (n – битовая длина хэш-кода).

4. Защищенность от прообразов второго рода: по данному сообщению сложно найти другое сообщение, имеющее такой же хэш-код.

5. Лавинный эффект: два сообщения, отличающиеся только одним битом, должны иметь сильно различающиеся хэш-коды. Иными словами, изменение одного бита приводит к лавинообразному изменению хэш-кода.

6. Случайное отображение: хэш-функция должна обладать свойствами равномерного случайного отображения, т.е. хэш-код можно рассматривать как псевдослучайное число с равномерным распределением.

Описание алгоритма хэширования MD4

Одним из способов построение хэш-функции является использование симметричного блочного шифра. Главным недостатком такого подхода является низкая скорость работы, поэтому на практике используют специально спроектированные эффективные алгоритмы хэширования. В качестве примера рассмотрим алгоритм MD4, предложенный Рональдом Ривестом в 1990 году [1].

Алгоритм MD4 генерирует 128-битный хэш-код для произвольного входного сообщения. Входное сообщение представляет собой поток битов (или байтов), и может иметь произвольную *битовую* длину (в том числе нулевую), которую обозначим буквой b.

Словом будем называть 32-разрядное целое число (unsigned int в языках C/C++). Слово рассматривается как группа из четырёх байтов, в которой младший байт является первым байтом группы (располагается в памяти по меньшему адресу). Например, группе из четырёх байтов (0x6F, 0x02, 0xE5, 0x8C) соответствует слово со значением 0x8CE5026F (префикс 0x обозначает число в 16-ричной системе счисления). Такой порядок байтов является естественным для большинства типов ЭВМ. Поэтому, если рассматривать слово как массив из 4-х байтов, то нулевой элемент этого массива будет равен 0x6F.

Блоком будем называть последовательность из 512 битов (16 слов). Исходное сообщение разбивается на блоки длиной 512 бит, каждый из которых последовательно подвергается процедуре хэширования. Результат хэширования предыдущего блока используется при хэшировании следующего блока. Перед хэшированием сообщение расширяется так, чтобы его битовая длина была кратной 512. Расширение сообщения состоит из двух этапов.

На **первом этапе** к сообщению добавляется единичный бит «1». После этого к нему добавляется необходимое число нулевых битов, чтобы длина сообщения была равна 448 по модулю 512. Единичный бит добавляется всегда, даже если длина сообщения уже равна 448 по модулю 512.

На **втором этапе** к сообщению добавляется 64-битное представление числа b (*битовой* длины исходного сообщения), в результате чего длина рас-

ширенного сообщения будет кратной 512. При этом первое 32-битное слово содержит младшую часть b, а второе слово – старшую (второе слово равно нулю, если длина сообщения не превосходит 2^{24} байт). Например, если сообщение состоит из трех *байтов*, то $b = 3*8 = 24$. Тогда первое 32-битовое слово будет равно 00000000’00000000’00000000’00011000, а второе слово будет состоять из одних нулей.

Для вычисления хэш-кода используется буфер, состоящий из 4-х слов: A, B, C, D. Вначале им присваиваются следующие шестнадцатеричные значения: A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476.

Произвольный блок сообщения можно представить в виде массива X, состоящего из 16 слов (X[0]…X[15]). Для каждого такого блока выполняются 3 раунда преобразований. Введём следующие обозначения.

Для *первого раунда* выражение [abcd k s] обозначает операцию

$$a = (a + H(b, c, d) + X[k]) <<< s.$$

Для *второго раунда* выражение [abcd k s] обозначает операцию

$$a = (a + G(b, c, d) + X[k] + 0x5A827999) <<< s.$$

Для *третьего раунда* выражение [abcd k s] обозначает операцию

$$a = (a + H(b, c, d) + X[k] + 0x6ED9EBA1) <<< s.$$

Выражение $x <<< s$ обозначает циклический сдвиг влево на s битов. В отличие от простого сдвига, при циклическом сдвиге влево старшие биты не теряются, а циклически переходят в начало слова x. Например, (01000111 <<< 2) = 00011101.

Вспомогательные функции F, G, H имеют следующий вид:

$$F(x, y, z) = xy \vee \bar{x}z$$
$$G(x, y, z) = xy \vee xz \vee yz$$
$$H(x, y, z) = x \oplus y \oplus z$$

В этих функциях используются поразрядные (битовые) операции: запись xy обозначает побитовое умножение x и y (конъюнкция, операция & в языке Си), $x \vee y$ обозначает побитовое сложение (дизъюнкция, операция | в языке Си),

$\bar{x}$ обозначает побитовое отрицание (инверсия, операция ~ в языке Си), $x \oplus y$ обозначает побитовое сложение по модулю два (исключающее «ИЛИ», операция ^ в языке Си).

Пусть расширенное сообщение представлено в виде массива *R*, составленного из *N* блоков. Тогда алгоритм вычисления хэш-кода можно записать в виде следующего псевдокода.

```
for i = 1 to N do begin
      X = R[i] // записываем в X очередной 256-битный блок
      // сохраняем начальные значения A, B, C, D
      AA = A
      BB = B
      CC = C
      DD = D
      // 1-й раунд. Порядок выполнения слева направо, сверху вниз.
      // То есть сначала первая строка, затем вторая строка и т.д.
      [ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
      [ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
      [ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
      [ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]
      // 2-й раунд
      [ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
      [ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
      [ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
      [ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]
      // 3-й раунд
      [ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
      [ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
      [ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
      [ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]

      A = A + AA
      B = B + BB
      C = C + CC
      D = D + DD
end
```

По окончании цикла хэш-код будет содержаться в переменных A, B, C, D, значения которых выводятся друг за другом в 16-ричном виде.

Примеры хэш-кодов для разных строк

MD4("") = 31d6cfe0d16ae931b73c59d7e0c089c0

MD4("abc") = a448017aaf21d8525fc10ae87aa6729d

MD4("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") = 043f8582f241db351ce627e153e7f0e4

Задания по лабораторной работе №4

Задание 1. Написать программу, вычисляющую с помощью алгоритма MD4 хэш-код строки, введённой пользователем.

Задание 2. Убедиться в наличии лавинного эффекта. Для этого в исходной символьной строке M изменить один произвольный бит и получить строку M'. Вычислить и сравнить хэш-коды $h = H(M)$ и $h' = H(M')$. Определить, какое количество битов в h' изменилось по сравнению с h. Для этого удобно использовать операцию сложения по модулю два. Число различных битов в h и h' будет равно числу единичных битов в $h \oplus h'$.

Задание 3. Написать программу, реализующую процедуру поиска коллизий алгоритма MD4, то есть двух строк, имеющих одинаковый хэш-код. Для сокращения числа операций ($\sim 2^n$) до разумного количества использовать не весь хэш-код, а лишь его часть. Пусть алгоритм MD4 возвращает хэш-код в виде четырёх слов A, B, C, D. Тогда в качестве хэш-кода будем использовать только младшие k битов слова A (т.е. k не должно превышать 32). Число k вводится пользователем.

Поиск коллизии производится следующим образом. Последовательно генерируются псевдослучайные строки фиксированной длины L (задаётся пользователем) и вычисляются их хэш-коды. Для хранения результатов этих операций использовать 2 массива: в один массив записываются сгенерированные строки, а во второй – соответствующие им хэш-коды. Указанные действия продолжать до обнаружения двух одинаковых элементов в массиве хэш-кодов. Вывести на экран соответствующие строки – обнаруженную коллизию, их общий хэш-код, а также общее количество сгенерированных строк N. Построить график зависимости $N(k)$.

Задание 4. Программно реализовать поиск прообраза для заданного хэш-кода некоторой строки. В качестве хэш-кода также использовать только часть реального хэш-кода (см. Задание 3). Исходным сообщением является парольная фраза (вводится пользователем), для которой вычисляется частичный хэш-код длиной k младших бит слова A (k задаётся пользователем). Путем формирования псевдослучайных строк найти строку, дающую тот же самый частичный хэш-код. Вывести найденную строку и количество потребовавшихся итераций N. В задании 4 при генерировании строк использовать только печатные символы (латинские буквы, цифры, знаки препинания и т.д.). Построить график $N(k)$.

Контрольные вопросы

1. Что такое хэш-функции? Для чего они применяются?
2. Что такое односторонность хэш-фукнции?
3. Что такое коллизия?
4. Что такое прообраз второго рода?
5. Что такое «лавинный эффект»?

ПРИЛОЖЕНИЕ

Операции с двоичным представлением целых чисел

Во многих криптографических алгоритмах шифруемые сообщения рассматриваются как набор битов, над которыми производятся некоторые операции. В этом приложении рассматриваются основные способы работы с двоичным представлением чисел, а также некоторые конкретные задачи, встречающиеся при выполнении лабораторных работ.

Все данные в памяти компьютера хранятся в двоичной системе счисления, однако непосредственный доступ к отдельным битам обычно невозможен. При необходимости получить доступ к отдельным битам используют поразрядные (битовые) операции. Некоторые из этих операций похожи на логические операции (логическое И, логическое ИЛИ и др.), однако в отличие от них выполняются для каждого бита числа. Побитовые операции могут выполняться над различными видами целых чисел: 8-битными, 16-битными, 32-битными, 64-битными. В примерах для простоты будут использоваться 8-битные числа. Нумерация битов производится справа налево начиная с нуля. Крайний правый бит является нулевым и называется *младшим*. Крайний левый бит называется *старшим*. Рассмотрим основные поразрядные операции.

Операция «побитовое И»

Эта бинарная операция представляет собой *конъюнкцию* (*логическое умножение*) битов двух целых чисел. Операция выполняется для каждой пары битов, стоящих в одинаковых разрядах. В языке Си она обозначается символом **&**. Обычно побитовое «И» используется для обнуления некоторых двоичных разрядов. Например, обнулим нулевой и второй разряды (считая справа):

```
  01101101
& 
  11111010
----------
  01101000
```

Видно, что результат равен 1 только если оба бита равны 1 (аналог умножения).

Операция «побитовое ИЛИ»

Это бинарная операция, которая в языке Си обозначается оператором |, представляет собой *дизъюнкцию* (*логическое сложение*) разрядов двух целых чисел. Используется для установки в единицу некоторых разрядов. Например:

```
  01101101
| 
  00000110
----------
  01101111
```

Если хотя бы один из двух битов не равен нулю, то результат равен 1.

Операция «побитовое исключающее ИЛИ» (*сложение по модулю* 2)

В языке Си записывается с помощью оператора ^. Например, *x* ^ *y* выполняет побитовое сложение по модулю 2 всех битов чисел *x* и *y*. Результат этой операции равен 0, если оба бита одинаковые, и равен 1, если они различны.

Операция «побитовое НЕ»

Эта унарная операция также называется *побитовым отрицанием* или *инверсией* и обозначается в языке Си оператором ~. Она инвертирует все биты числа, то есть 0 переходит в 1 и наоборот.

Поразрядный сдвиг влево

В языке Си обозначается как *x* << *n*. Все биты целого числа *x* сдвигаются влево на *n* битов. При этом левые *n* битов отбрасываются, а освободившееся справа место заполняется нулевыми битами. Например, (10011001 << 2) = 01100100. Эту операцию удобно использовать в сочетании с другими побитовыми операциями. Например, чтобы установить в единицу 6-й бит числа *x*, используется запись *x* | (1 << 6), а для обнуления 11-го и 15-го битов надо записать x & ~(1 << 11 | 1 << 15). В качестве упражнения рекомендуется записать двоичное значение выражения справа от знака &.

Поразрядный сдвиг вправо

В языке Си записывается как *x* >> *n*. Сдвигает все биты целого числа *x* на *n* битов вправо. Освободившиеся слева биты заполняются нулями (если старший бит *x* был равен 0 до сдвига) либо единицами (в противном случае). Если переменная *x* имеет беззнаковый тип (unsigned), то всегда заполняется нулями.

Операции сдвига позволяют выполнять действия над разными битами одного числа. Например, в Лабораторной работе №2 требуется выполнять сложение по модулю 2 некоторых разрядов регистра сдвига (пусть это будут 4-й и 18-й разряды, считая с нуля). Для этого выполним код (где *r* – регистр сдвига)

```
bit4 = ((1 << 4) & r) >> 4;
bit18 = ((1 << 18) & r) >> 18;
y = bit4 ^ bit18;
```

Осталось записать полученный бит *y* в начало регистра *r*, используя сдвиг влево.

ЛИТЕРАТУРА

1. Смарт Н. Криптография. М.: Техносфера, 2005. 528 с.
2. Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. М.: Гелиос АРВ, 2002. 480 с.
3. Карпов А.В. Компьютерное имитационное моделирование. Учебное пособие для магистрантов и студентов старших курсов. Казань: Казан. ун., 2004. 79 с.
4. Ишмухаметов Ш.Т. Методы факторизации натуральных чисел. Казань: Казан. ун., 2011. 190 с.
5. Шеннон Р. Имитационное моделирование систем – искусство и наука: Перевод с англ. – М.: Мир, 1978. – 418с.
6. Петраков А.В. Основы практической защиты информации. – М.: Радио и связь, 2000. – 362с.
7. Романец Ю.В., Тимофеев П.А. Защиты информации в компьютерных системах и сетях. – М.: Радио и связь, 2000. – 320с.