ИНСТИТУТ ФИЗИКИ
КАЗАНСКОГО (ПРИВОЛЖСКОГО) ФЕДЕРАЛЬНОГО
УНИВЕРСИТЕТА

КАРПОВ А.В., СУЛИМОВ А.И.

ВВЕДЕНИЕ В КРИПТОГРАФИЮ

(учебно-методическое пособие
по выполнению лабораторных работ)

Казань – 2022

Печатается по решению Учебно-методической комиссии Института физики Казанского (Приволжского) Федерального Университета
Протокол № ___ от «____» _____ 2022 г.

УДК 004.056.55, 003.26.09, 004.421.5

Карпов А.В.,Сулимов А.И. ВВЕДЕНИЕ В КРИПТОГРАФИЮ. Учебно-методическое пособие по выполнению лабораторных работ для магистрантов и студентов старших курсов. Казань, 2022. 57 с.

Издание второе (переработанное). Изложены основы защиты информации криптографическими методами, статистические методы проверки качества ключевых последовательностей, а также рассмотрены некоторые актуальные проблемы современной криптографии. 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 10.03.01 – Информационная безопасность (Безопасность автоматизированных систем) по дисциплине «Криптографические методы защиты информации» и обучающихся по специальности 03.04.03 – Радиофизика (Распределенные интеллектуальные системы) по дисциплине «Криптографические методы в распределенных интеллектуальных системах».

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. Каждая лабораторная работа содержит различные по сложности типы заданий. Большинство заданий имеют базовую сложность. Выполнение этих заданий гарантирует освоение изучаемого материала. Для различных направлений подготовки студентов предусмотрено выполнение заданий (стандартной сложности) средней сложности и повышенной сложности (*). Выполнение этих заданий стимулирует углубленное изучение рассматриваемой темы. Каждый раздел пособия заканчивается списком контрольных вопросов, которые позволяют студенту проверить усвоение теории и выделить для себя самое важное.

По результатам выполнения каждой лабораторной работы составляется отчёт, который представляется в виде электронного документа в формате doc 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 «Приложения».

Исторические шифры

Лабораторная работа №1

Реализация и криптоанализ шифра замены

Шифр замены является одним из самых первых известных типов шифров [1,2]. В простейшем варианте, 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. Для русского языка считается, что буква «ё» совпадает с буквой «е».

Нормативный алфавит, кроме непосредственно букв, может содержать и другие символы (пробел, цифры, знаки препинания и пр.). Символы, не входящие в нормативный алфавит, не подвергаются замене. Замена осуществляется по таблице, которая выполняет роль ключа шифра. Эта таблица задает порядок замены символов: каждый символ из левой части таблицы заменяется соответствующим символом из правой части таблицы. Ниже показан фрагмент возможной таблицы замены.

А	Е
Б	Р
В	Й
Г	О
…	…
Я	Б
пробел	П

Таблица замены работает симметрично в обе стороны, то есть отображение является взаимно однозначным. Это позволяет использовать ту же самую таблицу для дешифрования 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 различной частотой. Так, в русском языке наиболее распространенной является буква «О», а самой редкой буква – «Ф». На основе анализа большого числа текстов были найдены характерные частоты встречаемости всех букв. Будем рассматривать эти частоты как эталонные и обозначать символом q_s. Ниже приведена таблица, в которой буквы расположены в порядке убывания их частоты q_S.

Символ	Частота q_S	Символ	Частота q_S	Символ	Частота q_S
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	Б,	0,014	Э	0,003
В	0,038	Г	0,013	Ъ	0,002
Л	0,035	Ь	0,012	Ф	0,002

Рассмотрим, как реализуется статистический способ криптоанализа (взлома шифра) текстового сообщения. На ***первом шаге*** ста-

тистического частотного криптоанализа находятся эмпирические частоты встречаемости всех символов в зашифрованном сообщении. Эмпирическая частота символа x вычисляется по формуле:

$$q(x) = N(x) / N \qquad (1.1)$$

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 эмпирическую частотную таблицу шифротекста с априорной частотной таблицей русского языка.

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

Для получения дополнительной статистической информации анализируются частоты биграмм. *Биграммами* 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, можно рассмотреть и наиболее распространенные *триграммы* (последовательности из трех букв). В русском языке распространены триграммы: «СТО», «ЕНИ», «ОГО», «ОСТ», «ЧТО». Полезную информацию дают и 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 побуквенно, с постоянным контролем промежуточного текста. В случае обнаружения ошибочной замены (например, получено несуществующее слово), необходимо скорректировать одну или несколько замен.

Кроме истинного ключа, который используется при создании конкретной шифрограммы, необходимо иметь ввиду наличие ложных ключей. Ложный ключ – это любой возможный вариант ключа, отличный от истинного. В дальнейшем, мы будем использовать такую характеристику зашифрованного текста как *расстояние единственности*. Расстоянием единственности шифра называют такую длину шифротекста n_0 , начиная с которой среднее число ложных ключей обращается в ноль. Иными словами, если криптоаналитик перехватит более n_0 символов шифротекста, это со 100%-вероятностью позволит (например, полным перебором вариантов) восстановить исходный открытый текст. Если перехватывается менее n_0 символов, восстановление текста будет неоднозначным (могут быть несколько разных вариантов открытого текста).

С теоретико-информационной точки зрения, качество шифра определяется *энтропией* шифротекста $H(C)$. Согласно определению К. Шеннона [8], энтропия является мерой неопределенности элементов сообщения и характеризует среднее количество информации, передаваемое одной буквой. У идеального шифра энтропия $H(C)$ должна достигать теоретического максимума, что соответствует максимальной неопределенности криптоаналитика при его анализе. Слабой стороной шифра моноалфавитной замены является сохранение энтропии исходного естественного языка: $H(C)=H(M)$.

Связано это с тем, что естественный язык (в отличие от машинного) очень избыточный, так как между отдельными буквами имеется сильная статистическая связь. *Избыточностью* языка называется величина:

$$R = 1 - \frac{H(M)}{H_{\max}}, \qquad (1.2)$$

где $H_{\max}$ – теоретически максимальная энтропия ансамбля из n букв, а фактическая энтропия $H(M)$:

$$H(M) = -\sum_{i=1}^{n} p_i \log_2 p_i \quad [бит/симв.], \qquad (1.3)$$

при этом p_i – априорная (эталонная) вероятность возникновения i-й буквы. Очевидно, что для текста длиной N символов эмпирическая частота заданной буквы $q_i = N \cdot p_i$.

В теории доказывается [8], что для ансамбля n букв максимальная энтропия достигается при равновероятном законе распределения и составляет $H_{\max} = \log_2 n$. Например, для модельного русского языка (в котором буква «ё» заменяется на «е», буква «ъ» – буквой «ь», а также имеется пробел «_») $H_{\max} = \log_2 32 = 5$ *бит/симв.*, для английского языка – $H_{\max} = \log_2 26 = 4{,}7$ *бит/симв.*

Энтропию естественного языка можно определить следующим образом [9]:

$$H(M) = \lim_{n\to\infty} \frac{H("x_1x_2...x_n")}{n} = \lim_{n\to\infty} H("x_1x_2...x_n"|"x_1x_2...x_{n-1}"), \qquad (1.4)$$

где $H("x_1x_2...x_n")$ – энтропия n-граммы $"x_1x_2...x_n"$, $H("x_1x_2...x_n"|"x_1x_2...x_{n-1}")$ – условная энтропия буквы «x_n» при известных (n-1) предыдущих буквах.

Если бы все буквы были равновероятны, то избыточность языка была нулевой (R = 0). Если учесть неравновероятность появления букв, то энтропия русского языка снижается до H = 4,35 *бит/симв* (для английского языка до H = 4,14 *бит/симв*). При этом избыточность русского языка становится ненулевой: R = 0,13 (в случае английского языка – R = 0,12). Если далее учесть статистическую связь букв в пределах биграмм, триграмм и произвольных n-грамм, то энтропия снижается до $H(M)$ = 1,37 *бит/симв*. (для английского языка

– до $H(M)$=1,5 *бит/симв.*). Таким образом, согласно (1.3), избыточность русского языка R = 0,726, а английского языка – R = 0,681.

Расстояние единственности зависит от избыточности языка. В выражении (1.5) представлено его приближенное значение (нижняя оценка расстояния единственности)

$$N_0 \approx \frac{\log_2 K}{R \cdot \log_2 n}, \qquad (1.5)$$

где K – объем пространства ключей (для русского языка K = 32!, а для английского языка K=26!, соответственно).

Отсюда следует, что в случае, когда избыточность равна R=0 (идеальный шифр), расшифровке поддается только очень большой текст по размеру (близкий к бесконечному). Далее, если учесть неравновероятность появления букв, то для расшифровки достаточен текст длиной 189 символов (157 символов в случае английского языка). Если мы, кроме того, сможем учесть структуру и статистику языка в целом, то оказывается, что, теоретически, расшифровке поддаётся текст длиной всего 34 символа (для англоязычного текста – минимальная длина 28 символов). С учетом того, что средняя длина слова в русском языке около 4,7 букв (около 4 букв – в английском языке), для взлома шифра моноалфавитной замены, теоретически, достаточно всего одного предложения из семи слов.

Как зависит сложность расшифровки от длины текста? При проведении статистического анализа замечено, что чем короче текст, тем более заметны отличия наблюдаемых частот от эталонных. С увеличением объема текста шансы криптоаналитика на быструю и однозначную расшифровку возрастают. Связано это с тем, что с увеличением объема текста наблюдаемые частоты появления букв стремятся к эталонным значениям, что облегчает работу криптоаналитика на первом шаге статистического анализа.

Существует большое количество усовершенствованных модификаций шифра замены [6,7], усложняющих таблицу преобразования текста. Например, таблица может предусматривать замену одной *n*-граммы другой *m*-граммой. Такой метод существенно снижает избыточность текста *R* путем равномеризации распределения младших *n*-грамм. Тем не менее, остаточная статистическая связь более старших *n*-грамм, не позволяет снизить *R* до приемлемо низких значений, а дальнейшие способы усложнения шифра делают его применение непрактичным. В целом, можно заключить, что, по современным меркам, шифр замены не обеспечивает достаточной криптостойкости. Однако его изучение полезно для уяснения того, какими свойствами должен обладать качественный шифр.

Задания по лабораторной работе №1

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

- Шифрованию подвергается текст на русском языке, записанный в текстовом файле (в стандартной кодировке ОС Windows).
- Зашифрованный текст сохраняется в другом текстовом файле (encrypted.txt).
- Таблица замены (ключ шифрования) генерируется случайным образом и сохраняется в отдельный файл (key.txt).
- Расшифровка файла производится на основе таблицы, созданной в процессе шифрования (файл key.txt).
- Расшифрованный текст записывается в отдельный файл (decrypted.txt).

Задание 2. Расшифровать текст, выданный преподавателем, с помощью частотного криптоанализа (т.е. при отсутствии ключа шифрования). Текст получен путем замены букв и пробелов по неко-

торой (заранее неизвестной) таблице. Остальные символы (знаки препинания, цифры) остаются неизменными (не шифрованными).

В результате работы должна быть получена частотная таблица выданного зашифрованного текста, по формуле (1.3) должна быть вычислена энтропия текста (для этого априорные вероятности p_i нужно заменить нормированными эмпирическими частотами q_i/N), по формуле (1.2) соответствующая ей избыточность текста R, а также средняя длина слова: $n_{сл} = (N / q("\_")) - 1$, где $q(«\_»)$ – эмпирическая частота пробела. В отчете необходимо пояснить, как на основе построенной таблицы эмпирических частот проводилась расшифровка.

Преподаватель может предложить для расшифровки 1-2 текста. Объем текста:

1. Порядка 600 символов (задание повышенной сложности *).
2. Порядка 1000 символов (задание стандартной сложности).

Оформление отчёта

Результаты выполнения лабораторной работы должны быть оформлены в виде отчёта. Отчёт выполняется в виде электронного документа (в формате doc, docx или odt). Структура отчёта:

1. Название и номер работы, фамилия студента, номер группы.
2. Постановка задачи.
3. Описание процедуры шифрования и расшифрования.
4. Описание процедуры генерации таблицы замены.
5. Описание программы шифрования, демонстрационные скриншоты (изображения) работающей программы.
6. Выводы по первому заданию.
7. Зашифрованный текст, выданный преподавателем.
8. Частотная таблица зашифрованного текста.
9. Описание процедуры дешифрации текста.

10. Найденная таблица замены.
11. Расшифрованный текст.
12. Выводы по второму заданию.
13. В конце отчёта вставляется полный текст программы.

Контрольные вопросы

1. Какие исторические шифры вам известны?
2. Как осуществляется шифрование и расшифрование текста с помощью шифра замены?
3. Что такое ключ шифрования?
4. Сколько возможных ключей в шифре замены?
5. Легко ли подобрать ключ шифра замены методом прямого перебора?
6. В чем состоит уязвимость шифра замены?
7. Что такое частотная таблица и как она получается?
8. Как осуществляется частотный криптоанализ шифра замены?

Поточные шифры

Лабораторная работа №2

Генерация 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* (иногда – *гамма-последовательностью,* или просто *гаммой*). Ключевая последовательность произвольной длины формируется из короткого секретного ключа фиксированной длины согласно некоторому несекретному алгоритму.

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

$$шифротекст = текст \oplus ключ. \qquad (2.1)$$

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

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

$$(m \oplus k) \oplus k = m. \qquad (2.2)$$

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

Регистр сдвига с линейной обратной связью

Распространенный способ генерации ключевой последовательности основан на использовании регистра сдвига с обратной связью.

Он состоит из двух частей: ячеек памяти и цепи обратной связи. *Регистр сдвига* – это последовательность однобитовых ячеек памяти (триггеров) фиксированной длины, содержимое которых может сдвигаться в заданном направлении. Количество триггеров называется *длиной* регистра, или *разрядностью*.

Цепь обратной связи описывается некоторой булевой функцией от содержимого всех двоичных разрядов регистра и может принимать одно из двух значений: 0 или 1. Процесс генерации потока битов схематично показан на рисунке 2.1.

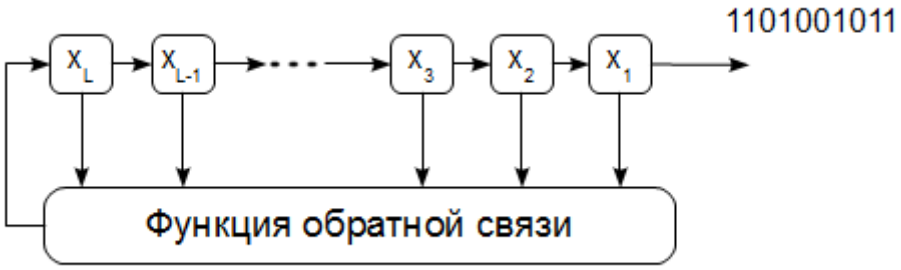

Рис. 2.1 – Структура сдвигового регистра с обратной связью.

Как видно из рис. 2.1, 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 элементом выходной двоичной последовательности.

Простейшим видом функции обратной связи является линейная функция, заданная в виде суммы по модулю 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; в противном случае $c_k = 0$.

Для 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. 2.2). Если начальным состоянием регистра является «1111», то на каждом такте его содержимое будет принимать значения: 1111, 0111, 1011, 0101, 1010, 1101, 0110, 0011, 1001, 0100, 0010, 0001, 1000, 1100, 1110, 1111, 0111, … 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.

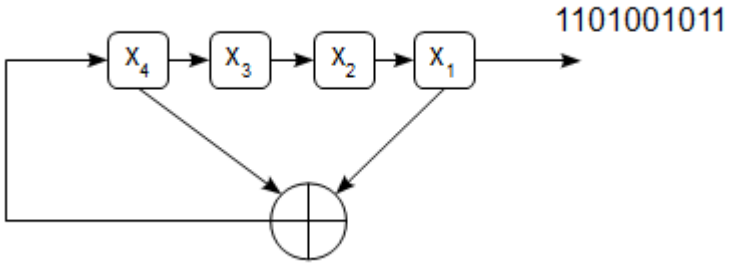

Рис. 2.2 – Пример работы сдвигового регистра с обратной связью.

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, спустя определенное количество тактов, генерируемая 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 максимальным периодом, называется *М-последовательностью*.

Чтобы выяснить условия, при которых LFSR будет обладать максимальным периодом, функции обратной связи ставят в соответствие полином $P(x) = c_L x^L + c_{L-1} x^{L-1} + \cdots + c_1 x + 1$. Так, регистру, приведенному выше в

качестве примера, соответствует полином $P(x) = x^4 + x^1 + 1$. Следует понимать, что полином $P(x)$ всего лишь в компактной символической форме описывает структуру цепи обратной связи и не используется для вычисления содержимого ячеек. Теоретический анализ показывает, что LFSR будет обладать максимальным периодом тогда и только тогда, когда полином $P(x)$ является *примитивным*. Неприводимый по модулю 2 полином называется примитивным, если в заданном числовом поле GF(2) не существует ни одного другого полинома $Q(x)$ порядка младше 2^L-1, делящегося без остатка на $P(x)$.

Ниже приведены некоторые примитивные полиномы, рекомендованные к применению на практике. Количество степенных слагаемых в полиноме соответствует числу отводов в цепи обратной связи регистра. В целях увеличения быстродействия процедуры генерирования ключевой последовательности желательно выбирать полиномы с наименьшим числом слагаемых, вида: $P(x) = x^L + x^S + 1$. Ниже приведены примеры таких полиномов:

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

Для повышения быстродействия программной реализации регистра его состояние выгодно хранить в виде целого L-разрядного числа, отдельные биты которого соответствуют ячейкам регистра. Для доступа к отдельным битам используются поразрядные операции (сдвиг, маскирование и пр.).

Недостатком рассмотренного генератора ключевой последовательность является его уязвимость к атаке на открытый текст. Если злоумышленник знает L идущих подряд битов последовательности, а также генераторный полином $P(x)$, он может однозначно предсказать всю остальную последовательность. Даже если генераторный полином неизвестен, то для его определения достаточно знать $2L$ битов последовательности, после чего структура полинома восстанавливается, например, с помощью алгоритма Берлекэмпа-Месси. Для уст-

ранения данной уязвимости на практике используют комбинации из нескольких генераторов с разными периодами.

Исследование статистических свойств M-последовательности

Двоичная последовательность, выдаваемая LFSR, должна удовлетворять определённым требованиям. С криптографической точки зрения, главным требованием является невозможность предсказать следующий бит последовательности при известных предыдущих битах, то есть статистическая независимость битов друг от друга.

Идеальная ключевая последовательность должна быть неотличима от *случайной равномерно распределённой* двоичной последовательности. Поскольку *M*-последовательность не является случайной, далее будем называть её *псевдослучайной* последовательностью. В статистике был разработан ряд критериев проверки гипотезы о случайности последовательности. На практике они реализуются в виде статистических тестов. Несмотря на многообразие тестов [10], в криптографии своеобразными стандартами стали наборы статистических тестов NIST и DIEHARD [11,12]. В частности, набор тестов NIST содержит 15 статистических тестов различной сложности и проводит весьма комплексную проверку статистических свойств двоичной последовательности. Следует отметить, что последовательность, порождаемая LSFR, отвергается тестами NIST ввиду недостаточной (линейной) сложности функции обратной связи.

Ниже мы рассмотрим только самые базовые статистические тесты, дающие общее представление о методике проверки свойств последовательности.

Проверка статистических гипотез

Тестирование сгенерированной ключевой последовательности на случайность является частным примером задачи проверки простой двухальтернативной статистической гипотезы. В данной задаче выдвигаются две взаимоисключающие гипотезы: H_0 и H_1. Гипотеза H_0 считается основной (более предпочтительной) и формулируется следующим образом:

***H*₀:** «Тестируемая выборка двоичных чисел $\{x_1,x_2,\ldots,x_M\}$ является фрагментом равномерно распределенной случайной последовательности. Отклонения её статистических свойств от эталонных объясняются лишь недостаточной репрезентативностью выборки».

Гипотеза H_1 является альтернативной (менее предпочтительной) и заключается в обратном предположении:

***H*₁:** «Тестируемая выборка не является фрагментом равномерно распределенной случайной последовательности. Наблюдаемые отклонения статистических свойств тестируемой выборки $\{x_1,x_2,\ldots,x_M\}$ от эталонных слишком значительны, чтобы объясняться малым объемом M».

Для обоснованного выбора одной из гипотез требуется привлечь статистический критерий. Критерий вводит *статистику* и оптимальное *правило принятия решений* по этой статистике, выбирающее одну из гипотез. Статистика задается в виде некоторой функции $V(x_1, x_2,..., x_M)$ от выборки. Конкретный вид формулы для расчета статистики V задается выбранным критерием.

После обработки выборки $\{x_1,x_2,\ldots,x_M\}$ критерий формирует численное значение статистики $v = V(x_1, x_2,..., x_M)$. Далее предполагают, что существует некая воображаемая выборка $\{x_{01}, x_{02},..., x_{0M}\}$ того же объема, которая является эталонным фрагментом равномерно распределенной случайной последовательности. Иными словами, воображаемая выборка $\{x_{01}, x_{02},..., x_{0M}\}$ идеально соответствует основной гипотезе H_0. Для этой воображаемой выборки вычисляют эталонное значение статистики: $v_0 = V(x_{01}, x_{02},..., x_{0M})$, после чего рассматривают *меру отклонения* от эталонного уровня $d = (v - v_0)$. Чем выше мера отклонения d для тестируемой выборки, тем менее правдоподобна гипотеза H_0 и более правдоподобна альтернативная гипотеза H_1.

Критерий принимает решение по следующему простому правилу:

$$\begin{cases} |\, v - v_0 \,| = |\, d \,| \le d_{кр}(\alpha) \rightarrow H_0; \\ |\, v - v_0 \,| = |\, d \,| > d_{кр}(\alpha) \rightarrow H_1. \end{cases} \qquad (2.3)$$

Величина $d_{KP}(\alpha)$ называется *критическим уровнем* отвержения основной гипотезы H_0, она вычисляется критерием по заданному *уровню значимости* α. В свою очередь, величина α определяет строгость критерия. С повышением α граница критической области $d_{KP}(\alpha)$ уменьшается, и критерий становится более «придирчивым» к тестируемой выборке. При высоких α даже малые отклонения $d = (v - v_0)$ от эталонного уровня считаются *значимыми* и достаточными для отвержения основной гипотезы H_0.

Теоретически, можно придумать неограниченное количество статистических критериев и сопряженных с ними статистик *V*. Качество критерия характеризуется вероятностями α и β ошибочных решений. Как бы ни был совершенен критерий, он все равно не застрахован от ошибок. При этом возможны ошибки двух типов: 1) ошибочное отвержение качественной выборки – ошибка I рода; 2) ошибочное подтверждение некачественной выборки – ошибка II рода. Вероятность ошибки II рода обозначается $\beta = P(|d| < d_{кр} \mid H_1)$. Запись "$....|H_1$" означает – при верности гипотезы H_1. Вероятность ошибки I рода обозначается $\alpha = P(|d| > d_{кр} \mid H_0)$ и часто называется *уровнем значимости* критерия.

Из последнего определения видно, что вероятность α необоснованно отвергнуть заведомо равномерную последовательность растет при уменьшении критического уровня d_{KP}. Если устремить $\alpha \to 0$, то статистический тест, наоборот, будет пропускать сколь угодно неравномерные и неслучайные выборки, так как при этом $d_{KP} \to \infty$. Величину (1-α) называют *доверительной вероятностью* теста. Следовательно, при α = 0 тест со 100%-вероятностью доверяет высокому качеству любой исследуемой ключевой последовательности, что приводит к абсолютной ненадежности теста, так как $\beta \to 1$.

Для создания надежного теста нужно обеспечить как можно меньшую вероятность пропуска ненадлежащей шифрующей последовательности: $\beta \to \min$. Иными словами, при заданном уровне значимости α требуется обеспечить максимальную *мощность критерия*: $(1-\beta) \to \max$. Таким образом, для надежного

тестирования ключевой последовательности необходимо привлекать наиболее мощные статистические критерии.

В последующих тестах будут использованы статистические критерии, обеспечивающие минимальный уровень вероятности ошибки II рода β при заданном уровне значимости α.

Сериальный тест

В сериальном тесте проверяется равномерность распределения *серий* (комбинаций) из нескольких последовательных элементов выборки. Например, рассмотрим серии из двух соседних битов. Всего возможны 4 различные серии: «00», «01», «10», «11». В равномерной случайной последовательности все комбинации возникают с равной вероятностью 1/4. В случае серии длиной k последовательных битов эталонная вероятность реализации составляет $p = \frac{1}{2^k}$.

При проведении сериального теста сгенерированная выборка из M битов разбивается на $N = \frac{M}{k}$ непересекающихся серий. Для каждой двоичной комбинации подсчитывается эмпирическая частота $N_j^Э$ (всего 2^k частот). Эталонная теоретическая частота каждой комбинации равна $N_j^T = \frac{N}{2^k} = \frac{M}{k \cdot 2^k}$.

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. Критерий χ^2 вводит следующую статистику:

$$V = \chi^2 = \sum_{j=1}^{2^k} \frac{(N_j^Э - N_j^T)^2}{N_j^T}. \qquad (2.4)$$

Из (2.4) очевидно, что для идеальной ключевой последовательности эталонный уровень статистики $v_0 = \chi_0^2 = 0$, поэтому мера отклонения численно совпадает со значением статистики: $d = \chi^2$.

Для заданного уровня значимости α определяется критический уровень $\chi_{кр}^2(\alpha, 2^k)$ такое, что выполняется условие

$$P(\chi^2 > \chi_{кр}^2(\alpha, 2^k)) = \alpha. \qquad (2.5)$$

Так как статистика (2.4) неотрицательна, то правило принятия решений (2.3) сводится к следующему:

$$\begin{cases} \chi^2 \le \chi_{кр}^2 \rightarrow H_0 - \textit{тест пройден}; \\ \chi^2 > \chi_{кр}^2 \rightarrow H_1 - \textit{тест провален}. \end{cases} \qquad (2.6)$$

Слишком малые значения χ^2 являются подозрительными. Они означают, что тестируемая последовательность равномерна, но, возможно, неслучайна – сгенерирована по специальному «эталонному» алгоритму. Таким образом, целесообразно задавать два уровня значимости, например, нижний α_{min} = 0,10 и верхний α_{max} = 0,9. Для каждого граничного уровня значимости вычисляется критический уровень $\chi_{кр}^2$, а правило принятия решений (2.6) модифицируется и приводится к виду:

$$\begin{cases} \chi^2 \in [\chi_{кр}^2(\alpha_{\max}); \chi_{кр}^2(\alpha_{\min})] \rightarrow H_0 - \textit{тест пройден}; \\ \chi^2 \notin [\chi_{кр}^2(\alpha_{\max}); \chi_{кр}^2(\alpha_{\min})] \rightarrow H_1 - \textit{тест провален}. \end{cases} \qquad (2.7)$$

В таблице 2.1 приведены критические уровни статистики χ^2 для различного числа степеней свободы (2^k – 1) и уровней значимости α.

Табл. 2.1 – Критические уровни статистики χ^2

Количество степеней свободы	Уровень значимости α					
	0,95	**0,9**	**0,8**	**0,2**	**0,1**	**0,05**
3 (*k* = 2)	0,352	0,584	1,005	4,640	6,251	7,815
6	1,645	2,200	3,070	8,560	10,640	12,590
7 (*k* = 3)	2,167	2,833	3,820	9,800	12,017	14,067
15 (*k* = 4)	7,261	8,547	10,310	19,310	22,307	24,996

Покер-тест

Данный тест позволяет обнаружить детерминированные структуры внутри ключевой последовательности, наличие которых повышает шансы криптоаналитика на успешный анализ.

При проверке покер-тестом выборка из M битов разбивается на $N=M/32$ непересекающихся 32-битных подпоследовательностей. Каждая из них преобразуется сначала в целое десятичное число X_i, а затем – в дробное число $r_i = \frac{X_i}{2^{32}-1}$, и наконец снова в целое десятичное число вида $u_i = \lfloor q \cdot r_i \rfloor$, где q – произвольное целое число. При реализации покер теста рекомендуется выбирать q = 10. После данного преобразования исходная двоичная выборка $\{x_1,x_2,\ldots,x_M\}$ преобразуется в выборку десятичных чисел $\{u_1,u_2,\ldots,u_N\}$.

Разобьем вторичную выборку $\{u_1,u_2,\ldots,u_N\}$ на $(N/5)$ квинтетов. Возможны семь классов квинтетов, отличающихся различным содержанием цифр (порядок появления цифр не имеет значения):

1. a,b,c,d,e – все цифры различны (например, 67834);
2. a,a,b,c,d – две цифры совпадают, остальные различны (например, 55490);
3. a,a,b,b,c – имеются два дуплета (например, 55499);

4. *a,a,a,b,c* – имеется триада, остальные две различны (например, 77743);

5. *a,a,a,b,b* – имеется триада и дуплет (например, 55599);

6. *a,a,a,a,b* – имеется тетрада (например, 88882);

7. *a,a,a,a.a.a* – совпадают все пять цифр (например, 77777).

Для идеальной ключевой последовательности эталонные вероятности каждой комбинации (для $q > 5$) вычисляются следующим образом

$$P_1 = \frac{(q-1)\cdot(q-2)\cdot(q-3)\cdot(q-4)}{q^4}, \quad (2.8)$$

$$P_2 = 10\cdot\frac{(q-1)\cdot(q-2)\cdot(q-3)}{q^4}, \quad (2.9)$$

$$P_3 = 15\cdot\frac{(q-1)\cdot(q-2)}{q^4}, \quad (2.10)$$

$$P_4 = 10\cdot\frac{(q-1)\cdot(q-2)}{q^4}, \quad (2.11)$$

$$P_5 = 10\cdot\frac{(q-1)}{q^4}, \quad (2.12)$$

$$P_6 = 5\cdot\frac{(q-1)}{q^4}, \quad (2.13)$$

$$P_7 = \frac{1}{q^4}, \quad (2.14)$$

В имитационном эксперименте определяется экспериментальное число появления каждой комбинации $N_j^э = p_j N/5 = p_j N/160, (j = \overline{1..7})$. Далее, как и в предыдущем тесте, проверка статистических гипотез осуществляется на основе критерия χ^2 (в данном случае, количество степеней свободы равно 6).

Корреляционный тест

В данном тесте анализируется статистическая связь между элементами последовательности x_i и x_{i+k} с использованием её автокорреляционной функции $R[k]$. Выборочная оценка элементов автокорреляционной функции (их также называют *коэффициентами автокорреляции*) по отрезку последовательности длиной N битов может быть вычислена по формуле:

$$R[k] = \frac{\frac{1}{N-k}\sum_{i=1}^{N-k}(x_i - m_i)\cdot(x_{i+k} - m_{i+k})}{\sqrt{D[x_i]\cdot D[x_{i+k}]}} \qquad (2.15)$$

где выборочные оценки математических ожиданий начального ($i = \overline{1, N-k}$) и конечного ($i = \overline{k+1, N}$) отрезков последовательности:

$$m_i = \frac{1}{N-k}\sum_{i=1}^{N-k} x_i, \qquad m_{i+k} = \frac{1}{N-k}\sum_{i=k+1}^{N} x_i, \qquad (2.16)$$

а соответствующие выборочные оценки дисперсий:

$$D[x_i] = \frac{1}{N-k-1}\sum_{i=1}^{N-k}(x_i - m_i)^2, \quad D[x_{i+k}] = \frac{1}{N-k-1}\sum_{i=k+1}^{N}(x_i - m_{i+k})^2, \quad (2.17)$$

Большое абсолютное значение $|R[k]|$ коэффициента автокорреляции указывает на сильную статистической связь между i-м и (i+k)-м членами последовательности, что недопустимо. Так как $R[k]$ – функция от выборки, то является статистикой. Для идеальной ключевой последовательности эталонное значение статистики $|R_0[k]| = 0$, $\forall k$, что указывает на независимость всех её элементов.

С учетом вышесказанного, соответствующее правило принятия решений в корреляционном тесте может быть записано в виде:

$$\begin{cases} |R[k]| \le R_{кр} \rightarrow H_0 - тест\ \ пройден; \\ |R[k]| > R_{кр} \rightarrow H_1 - тест\ \ провален. \end{cases} \qquad (2.18)$$

Для уровня значимости α=0,05 критический уровень статистики:

$$R_{кр}(\alpha = 0{,}05) = \frac{1}{N-1} + \frac{2}{N-2}\sqrt{\frac{N(N-3)}{N+1}} \qquad (2.19)$$

Для объективного тестирования необходимо рассмотреть уровень корреляции при различных значениях k = 1, 2, 3, ...

Методы повышения качества ключевой последовательности

В криптографии статистическим свойствам ключевых последовательностей придают большое значение. В серьёзных прикладных задачах уровень значимости α при тестировании последовательности значительно повышают. Вследствие этого, сгенерировать стандартными алгоритмическими, аппаратурными и физическими средствами ключевую последовательность удовлетворительного качества становится чрезвычайно сложно. Для повышения эффективности процесса генерации применяют специальный класс хэширующих преобразований, называемых *экстракторами случайности* [13,14].

Суть экстрактора случайности поясняется на рис. 2.3. На вход специального хэширующего алгоритма подаётся двоичная последовательность длины M с недостаточно удовлетворительными статистическими свойствами. На выходе формируется ε-равномерная случайная последовательность меньшего объема $N(\varepsilon) \le M$. Степень экстракции $N(\varepsilon)/M$ определяется желаемым приближением ε к эталонным свойствам ключевой последовательности. При $\varepsilon \to 0$ выходной эффект экстрактора также стремится к нулю: $N(0) \to 0$, поэтому на практике ищут некоторое компромиссное решение между качеством и количеством.

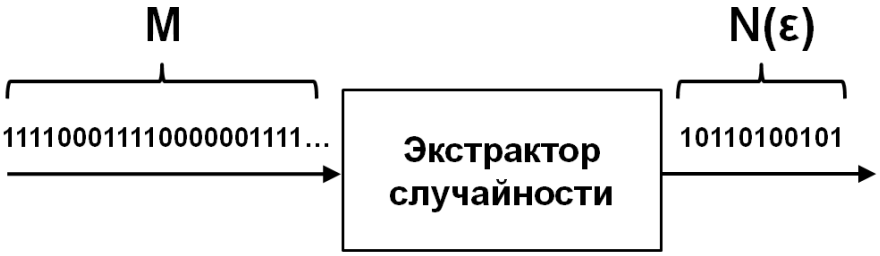

Рис. 2.3 Принцип работы экстрактора случайности

Мощь экстракторов случайности заключается в том, что они, теоретически, способны скорректировать статистические свойства ключевой последовательности, сгенерированной любым способом. Особым семейством экстракто-

ров случайности являются алгоритмы усиления секретности, которые обсуждаются в Лабораторной работе №4.

Задания по лабораторной работе №2

Задание 1. Написать программу, реализующую генератор M – последовательности на основе полинома обратной связи. Длина регистра L задается преподавателем. Начальные значения битов регистра сдвига задаются случайным образом (с помощью стандартного генератора псевдослучайных чисел). В программе должна быть предусмотрена возможность сохранения начального состояния регистра сдвига в файле key.txt, а также возможность загружать начальное состояние регистра из этого же файла. Предусмотреть в программе вывод первых N двоичных элементов M-последовательности на экран.

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

Задание 2. Протестировать M-последовательность длиной N бит (N задаётся преподавателем):

2.1. с помощью сериального теста. Использовать серии длиной k = 2, 3 или 4 бита. Вычислить статистику χ^2 по формуле (2.4) и реализовать правило принятия решений (2.6) или (2.7) для заданного уровня значимости α.

2.2(*). с помощью покер теста. Вычислить статистику χ^2 по формуле (2.4) и реализовать правило принятия решений (2.6) для заданного уровня значимости α.

2.3. Протестировать M-последовательность с помощью корреляционного теста. Вычислить значение $R[k]$ по формуле (2.15) для k = 1, 2, 8, 9.

Задание 3. 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 файл encrypted.txt. Расшифровать файл encrypted.txt с помощью того же ключа и убедиться, что расшифрованный и исходный файлы идентичны. Шифрование и расшифрование осуществляется по формуле (2.1)

путём сложения по модулю два (операция «исключающее ИЛИ») двоичного представления шифруемого файла с *M*-последовательностью.

Задание 4. Протестировать исходный (открытый) текст с помощью статистических тестов из задания 2. Объяснить результаты тестирования.

Задание 5. Протестировать шифрограмму с помощью статистических тестов из задания 2. Объяснить результаты тестирования.

Задание 6(*). Уменьшить размер ключевой последовательности до L = 8-16 бит и выполнить с её помощью шифрование информационного файла. Для этого использовать периодическое дополнение ключевой последовательности до размера файла M битов путем повторения ключевой последовательности (M/L) раз. Для зашифрованного файла построить график автокорреляционной функции $R[k]$ с глубиной статистической связи $k = 0,1,2,\ldots,2L$. Объяснить полученный график.

Контрольные вопросы

1. Что такое поточный шифр?
2. Как осуществляется поточное шифрование?
3. Как осуществляется поточное расшифрование?
4. Что такое ключевая последовательность (шифрующая гамма)?
5. Каким образом формируется ключевая последовательность?
6. Как работает регистр сдвига с линейной обратной связью?
7. Что такое *M*-последовательность? Каковы условия её генерации?
8. Какими свойствами должна обладать шифрующая ключевая последовательность?
9. Как проводится тестирование свойств ключевой последовательности?
10. Для чего и как используется статистический критерий χ^2?
11. Что такое коэффициент автокорреляции? Какие он может принимать значения?
12. Как можно улучшить статистические свойства ключевой последовательности?

Криптосистемы с открытым ключом

Лабораторная работа №3

Асимметричное шифрование на основе алгоритма RSA

Асимметричную схему шифрования иначе называют шифрованием с открытым ключом. Данная схема оперирует двумя ключами шифрования. Первая часть, называемая *открытым ключом* (с англ. – public key), используется для шифрования данных, а вторая часть (*закрытый ключ*, иногда – *секретный ключ,* с англ. – private key) используется для их расшифрования. Закрытый ключ хранится получателем сообщения в секрете. В то же время открытый ключ свободно передаётся по открытым каналам связи, а значит, кто угодно может использовать его для шифрования данных.

Надёжность асимметричных алгоритмов основана на *вычислительной сложности* нахождения закрытого ключа по известному открытому ключу шифрования. В криптографии с открытым ключом используют так называемые односторонние функции с лазейкой (с англ. – trapdoor one-way functions), которые легко вычисляются при прямом отображении аргумента, а обратная функция вычисляется за малое (полиномиальное) время, только если известна «лазейка», то есть некоторые дополнительные сведения о структуре функции. В отсутствие этих сведений, обращение функции становится вычислительно сложной задачей.

Может показаться, что наличие лазейки несет критическую угрозу безопасности криптосистемы и является негативным фактором. Однако лазейка абсолютно необходима для обеспечения удобства расшифровки информации легальными пользователями. Лазейкой в криптосистемах с отрытым ключом, как правило, является закрытый ключ, при знании которого легальные пользователи легко восстанавливают исходное сообщение. Если же исключить из криптографического алгоритма лазейку, то легальным пользователям для расшифровки сообщения придётся решать столь же сложную задачу, что и криптоаналитику.

Исторически первым и наиболее распространённым асимметричным шифром является алгоритм *RSA*. Его надежность основана на вычислительной сложности задачи *факторизации* (разложения целого числа на простые множители). В то же время прямая задача (нахождение числа по его простым множителям) является тривиальной. Обсудим далее процедуру генерации пары ключей, а также процедуру шифрования/расшифрования данных.

Процедура генерации открытого и закрытого ключей RSA

1. Случайно выбираются два простых числа p и q заданной битовой длины $L/2$, где величину L называют *длиной ключа*.
2. Вычисляется их произведение $n=p\cdot q$, которое называется *модулем шифрования.*
3. Вычисляется функция Эйлера от значения модуля: $\varphi(n) = (p-1)\cdot(q-1)$.
4. Выбирается целое число e, взаимно простое с $\varphi(n)$, то есть такое, что *НОД*$(\varphi(n)\ ,\ e) = 1$. Число e называется *открытой экспонентой*. Для ускорения операций шифрования в качестве открытой экспоненты берут простые числа, содержащие малое число единиц в двоичной записи (например, *числа Ферма*: 3, 17, 257, 65537). НОД (наибольший общий делитель) вычисляется согласно алгоритму Евклида (см. Приложение).
5. **Пару чисел (e, *n*) используют в качестве открытого ключа.**
6. Для создания закрытого ключа вычисляется число d, удовлетворяющее условию $(e\cdot d) \bmod \varphi(n) = 1$. Для вычисления d (*закрытой экспоненты*) используется расширенный алгоритм Евклида (см. Приложение). 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, не превышающие *n* (модуля шифрования). Обычно длину блоков выбирают примерно равной *L*/4. Каждый *i*-й блок шифруется по формуле $C_i = M_i^e \bmod n$. Для дешифрации *i*-го зашифрованного блока C_i используется формула $M_i = C_i^d \bmod n$.

Здесь 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 (а закрытая экспонента *d* имеет порядок модуля шифрования *n*) простое перемножение оказывается затратным, поэтому на практике используют алгоритм быстрого возведения в степень, позволяющий существенно сократить количество вычислительных операций (см. Приложение).

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
    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 mod N = 1, которое является единственным, если НОД(a, N) = 1. Результатом работы алгоритма будет разложение $1 = as + Nt$, которое иначе можно записать как as mod N = 1. Следовательно, коэффициент разложения s является искомым решением уравнения ax = 1 (mod N). Далее приводится псевдокод этого алгоритма.

```
входные данные: m, p
выходные данные: d, s, t
цель алгоритма: найти d=НОД(m, p) и найти s, t такие,
что d = s*m + t*p
начало алгоритма
a = m; b = p;
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 данными $m = e$ и $p = \varphi(n)$. 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, которое затем проверяется на простоту. Если сгенерированное число не является простым, процедура генерации повторяется, пока не будет получено простое число.

Чтобы сгенерировать случайную строку длиной L бит, она заполняется случайными битами, причём старший (L-й) и младший биты должны быть «1». Сгенерированная строка проверяется с помощью специальных вероятностных тестов (аналогичны проверке статистической гипотезы о простоте числа), поскольку достоверная проверка больших чисел занимает слишком много времени. Простейший вероятностный тест основан на следующей *теореме Ферма* (*тест Ферма*):

«если число T – простое, то для любого $a < T$ выполняется условие $a^{T-1} \bmod T = 1$».

Чем больше различных чисел a будет проверено на соответствие этому условию, тем достовернее тест (тем ниже вероятность β ошибки II рода при проверке гипотезы о простоте числа T). Если условие выполняется для k различных чисел a_k, то достоверность теста (мощность процедуры проверки 1-β) составляет $(1-2^{-k})$. Таким образом, процедура тестирования сводится к генерации k различных (возможно, случайных) чисел a_k, не превосходящих T, для каждого из которых проверяется условие Ферма. Чем больше k, тем меньше вероятность β ошибки II рода. Так, для $k = 10$ вероятность того, что непростое число будет ошибочно признано простым, составляет β=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, предпочитают использовать 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 менее 1024 бит. Значит, нам необходимо осуществлять арифметические операции над очень большими целыми числами, заведомо превышающими длину машинного слова. Стандартные процессоры позволяют непосредственно работать с числами длиной не более 64 бит. Чтобы обойти это ограничение, используются специальные алгоритмы, относящиеся к арифметике многократной точности.

В общем случае, реализация алгоритмов многократной точности является непростой задачей, поэтому на практике рекомендуется использовать готовые библиотеки. Для языка C++ могут быть рекомендованы библиотеки MPIR и BigInteger (последняя описана в Приложении). Некоторые языки, такие как Python, Java и Ruby, имеют встроенную реализацию арифметики многократной точности.

Библиотека BigInteger

Данная библиотека предоставляет удобные инструменты работы с *целыми* числами произвольной длины. Локализованные реализации библиотеки BigInteger существуют практически для всех популярных языков программирования. В качестве примера, рассмотрим её реализацию для языка C++.

Для представления больших 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, а также 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 фрагмент программного кода, иллюстрирующий подключение и использование библиотеки BigInteger.

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

int main()
{
    BigUnsigned x, y, z;
    std::string s;
    std::cin >> s; // ввод длинного числа в виде строки
    x = stringToBigUnsigned(s); // преобразование строки в число
    x *= 1000;
    z = x % stringToBigUnsigned("123456789"); // остаток от деления
    std::cout << "z = " << z << "\n";
    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 уже для относительно коротких ключей. Однако популярность алгоритма *RSA* стимулировала поиск новых эффективных методов факторизации. Одним из наиболее быстрых среди простых методов факторизации является метод ρ-эвристики Полларда. С вероятностью не менее 1/2, решение задачи находится за $\sim n^{1/4}$ итераций. Таким образом, данный метод относится к классу *экспоненциальных алгоритмов*. Более совершенные (*субэкспоненциальные*) алгоритмы факторизации (например, метод квадратичного решета, метод решета числового поля, метод эллиптических кривых и пр.) имеют гораздо более сложную реализацию, поэтому остановимся на методе ρ-эвристики.

Для факторизации числа n используется последовательность псевдослучайных чисел $\{x_i\}$, не превосходящих n. Получить такую последовательность можно, например, с помощью рекурсивной формулы: $x_{i+1} = (x_i^2 \pm 1) \bmod n$ (x_0 выбирается равным 0 или 2). 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 поиска делителя рассматриваются не все пары (x_i, x_j), а только те, для которых $j = 2^k$, а i пробегает значения от $2^k + 1$ до 2^{k+1} при различных k = 1, 2, 3…. Например, при k = 1 индекс j фиксируется на значении j = 2, а индекс i пробегает значения от 3 до 4. При k = 2 индекс j фиксируется на значении 4, а i варьируется от 5 до 8 (включительно) и так далее.

В заключение следует 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 n равна $2L(p)$ = 64 битам, а не $L(p) + L(q)$ =160. Поэтому на практике рекомендуется выбирать числа p и q одинаковой битовой длины.

Задания по лабораторной работе №3

Задание 1 (только для базового уровня)

- Получить от преподавателя 2 простых числа (p и q)
- Ввести произвольный текст (не менее 10 символов).
- Каждый символ (включая пробелы и знаки препинания) закодировать двузначным числом. Вывести на экран цифровой код введенного текста.
- Задать параметры: $n = pq$ *и* e .
- Вычислить d .
- Зашифровать сообщение, каждая буква шифруется по отдельности. Вывести на экран цифровой код шифрограммы.
- Расшифровать шифрограмму. Вывести на экран цифровой код. Вывести на экран восстановленный текст.

Задание 2. Программно реализовать процедуру генерации открытого и закрытого ключей заданной длины $L \geq 128$. В качестве открытой экспоненты использовать одно из чисел Ферма (3, 17, 257, 65537…). Открытый ключ в виде пары числе (e,n) записать в файл public.txt, Закрытый ключ в виде пары чисел (d,n) записать в файл private.txt.

Задание 3. Программно реализовать процедуры шифрования и дешифрования согласно алгоритму *RSA*. Считать открытый и закрытый ключи из файлов public.txt и private.txt.

На вход программы шифрования подается файл произвольного формата (достаточно рассмотреть лишь текстовые файлы), который разбивается на блоки длиной $K = L/4$ *бит* (где K должна быть кратно 8, чтобы блок содержал целое число байтов). Каждый K-битный блок представляется в виде целого числа M_i, которое шифруется по формуле $C_i = M_i^e \bmod n$. В результате шифрования блока M_i, получается число C_i, которое записывается в выходной файл encrypted.txt. В простейшей реализации, числа C_i могут храниться в отдельных

строках файла encrypted.txt в десятичном формате. В более совершенных реализациях, запись в файл encrypted.*** может осуществляться последовательным бинарным потоком с сохранением исходного расширения.

При расшифровывании файла encrypted.*** из него последовательно считываются шифр-блоки C_i (в общем случае, они могут иметь неодинаковые битовые длины, поэтому нужно продумать способ отделения одного шифр-блока от соседних). Каждый шифр-блок расшифровывается по формуле $M_i = C_i^d \bmod n$ и последовательно записывается в выходной файл decrypted.***. При правильной реализации процедур шифрования/расшифрования исходный и расшифрованный файл должны совпадать.

В простейшей реализации, при расшифровывании файла encrypted.txt из него считываются строки, содержащие десятичную запись целого числа. Это число расшифровывается с помощью закрытого ключа, разбивается на группы по 8 битов (или по 16 битов – для кодировки unicode) и записывается в выходной файл decrypted.txt в виде набора символов.

Задание 4. Проверить работоспособность алгоритма *RSA* в случае, когда одно из чисел p и q не является простым.

Задание 5. 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 для любого считанного из файла public.txt числа n.

Провести атаку для различных длин ключа, начиная с L = 70. Убедиться, что с ростом битовой длины ключа 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 35 бит с шагом 5 бит. Когда время расчетов превысит 1 минуту, уменьшить шаг до 1 бита.

Задание 6. Выяснить, как влияет различие битовой длины чисел p и q на сложность факторизации. Для этого берется максимальная длина ключа, найденная в Задании 5, которую обозначим буквой L^*. Затем генерируются простые числа p и q, имеющие длину rL^* и $(1–r)L^*$ соответственно (где $r < 1$). Построить график зависимости времени факторизации числа $n = p \cdot q$ от r для r = 0,25 … 0,5 с шагом 0,025.

Задание 7(*). Исследовать зависимость длительности $t_{enc}(V)$ шифрования и длительности $t_{dec}(V)$ дешифрования от битовой длины V сообщения при трёх различных длинах ключа L = {256; 512; 1024} бит. Результаты исследований представить в виде графиков: 3 кривые на одном рисунке для $t_{enc}(V)$ и 3 кривые на другом рисунке для $t_{dec}(V)$, соответственно. Объяснить полученные графики.

Задание 8(*). Исследовать зависимость коэффициента разрастания шифрограммы $k_V = V_Ш/V$ (где $V_Ш$ – битовая длина шифрограммы) от битовой длины сообщения V при трёх различных длинах ключа L = {256; 512; 1024} бит. Результаты исследований представить в виде графиков: 3 кривые на одном рисунке для $k_V(V)$.

Задание 9(*). Используя программы, разработанные в рамках Лабораторной работы №2 протестировать битовый поток шифрограммы на равномерность при двух значениях открытой экспоненты: e = 3 и e = 65537. Объяснить полученные результаты и сделать соответствующие выводы.

Контрольные вопросы

1. Дайте определение криптографической односторонней функции.
2. Приведите примеры односторонних криптографических функций.
3. Для чего используются два ключа? В чем их различие?
4. В чем преимущества асимметричных схем шифрования по сравнению с симметричными?

5. В чем недостатки асимметричных схем шифрования по сравнению с симметричными?
6. На чем основана надежность криптосистемы *RSA*? Почему при её реализации используются только простые числа?
7. Каково соотношение между объёмами исходного текста и шифрограммы? Чем это объясняется?
8. Каково соотношение между временами шифрования и расшифрования? Чем это объясняется?
9. Как реализуется разложение чисел на множители методом ρ-эвристики Полларда?
10. Как производится быстрое возведение в степень? Почему желательно иметь наименьшее количество единиц в двоичном представлении e?
11. На что влияет битовая длина открытой экспоненты e?
12. Объясните суть статистического подхода к генерации простых чисел.
13. 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) произвольного объёма и формирует на выходе двоичный код фиксированной длины, называемый *хэш-кодом* (или просто *хэшем*) сообщения. Хэш-функции находят широкое применение в теории кодирования, передачи информации и разработке программного обеспечения. Примером простейших хэш-кодов являются контрольные суммы и CRC-коды, которые предназначены для обнаружения случайных повреждений данных.

В криптографических приложениях к хэш-функциям предъявляются более жёсткие требования [1]. Функции, отвечающие этим требованиям, называются *криптографическими хэш-функциями*. Наиболее известными алгоритмами хэширования являются алгоритмы семейств MD, SHA и отечественный алгоритм Стрибог (ГОСТ 34.11-2018). Ниже приведены основные области применения криптографических хэш-функций.

- Проверка целостности сообщения.
- Электронная цифровая подпись и аутентификация.
- Хранение и проверка паролей.
- Выработка сильного ключа шифрования.

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 II рода* («ложным» прообразом).

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

Отличие прообраза II рода от коллизии заключается в том, что истинный прообраз (некоторое сообщение M) заранее фиксировано, 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). Это достигается заменой арифметических операций поразрядными операциями над машинными словами.

2. Односторонность: вычислительно сложно найти истинный прообраз M заданного хэш-кода h. Под вычислительной сложностью понимается отсутствие более эффективного алгоритма, чем прямой перебор (т.е. перебор различных сообщений до нахождения сообщения M' с заданным хэш-кодом h). Вычислительная сложность прямого перебора зависит от длины хэш-кода: если хэш-код имеет длину n бит, то для нахождения сообщения M' с таким хэш-кодом потребуется перебрать в среднем 2^n различных вариантов.

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 II рода: по заданному сообщению сложно найти другое сообщение, имеющее такой же хэш-код.

5. Лавинный эффект: два сообщения, отличающиеся только одним битом, должны иметь сильно отличающиеся хэш-коды. Иными словами, минимальное изменение сообщения должно лавинообразно изменять хэш-код.

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

Применение хэш-функций для усиления секретности

Одной из важных проблем в криптографии является обеспечение секретности криптосистемы при частичной утечке ключа. Для решения этой задачи был разработан специальный класс хэш-функций *усиления секретности* (с англ. – privacy amplification) [15]. Суть процедуры усиления секретности состоит в том, что из исходного частично скомпрометированного ключа K длиной L бит путем хэширования формируют безопасный ключ $k = G(K)$ меньшей длины n (см. рис. 4.1). Усиление секретности лежит в основе современных протоколов согласования ключей между абонентами [18,19], в том числе в системах квантовой криптографии [16]. Рассмотрим применение усиления секретности.

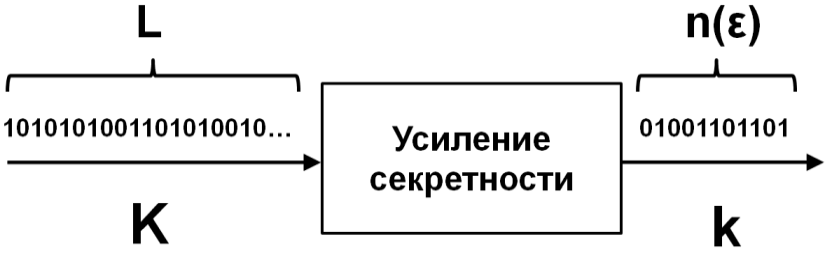

Рис. 4.1 – Принцип усиления секретности.

Допустим, криптоаналитик каким-либо образом получил знание о любых (не обязательно подряд идущих) C битах исходного ключа K. В лучшем случае (если между перехваченным фрагментом и остальными (L-C) битами нет статистической связи), это знание просто сокращает пространство ключей для перебора криптоаналитиком в 2^C раз. Однако в общем случае, это знание также порождает и некоторую неравномерность распределения в пространстве (L-C)-битных ключевых остатков, которая существенно сокращает время поиска ключа, так как некоторые варианты становятся более очевидными, чем иные.

Для устранения возникшей угрозы требуется сжать исходный ключ K до $n(\varepsilon) < (L - C)$ битов, избавившись от скомпрометированной ключевой информации. Размер $n(\varepsilon)$ выходного ключа k зависит от заданных требований секретности. Согласно этим требованиям, процедура усиления секретности формиру-

ет так называемый *ε-секретный ключ*, где ε – желаемая степень приближения к абсолютной секретности ключа *k*. Чем ниже ε задается, тем меньший объем *n*(ε) получается после хэширования исходного небезопасного ключа *K*.

Строгое математическое определение ε-секретности ключа даётся с использованием метрики среднего *вариационного расстояния* двух совместных распределений вероятностей:

$$d\left\{P_{kC}(k,C),\ 2^{-n}\cdot P(C)\right\}=\frac{1}{2}\sum_{c\in C}\sum_{k_i=0}^{2^n-1}\left|P_{kC}(k_i,c)-2^{-n}\cdot P(c)\right|\le\varepsilon. \qquad (4.1)$$

Критерий (4.1) требует, чтобы совместное распределение вероятностей $P_{kC}(k,C)$ итогового ключа k и перехваченной криптоаналитиком информации C сколь угодно мало (с точностью до ε) отличалось от совместного распределения $2^{-n}\cdot P(c)$ статистически независимых величин. При этом итоговый ключ k должен быть равновероятной строкой длиной n бит. Иными словами, перехваченная криптоаналитиком информация C не должна давать значимой информации об итоговом безопасном ключе k. Критерий (4.1) должен выполняться для любой возможной комбинации C битов, перехваченной криптоаналитиком.

Далеко не любая хэш-функция способна создать ключ, удовлетворяющий критерию ε-секретности (4.1). В теории доказывается [15], что для этого требуется использовать *универсальные хэш-функции второго порядка* (с англ. – two-universal hash functions).

Ввиду громоздкости, мы не будем приводить здесь строгое определение универсальной хэш-функции. Отметим лишь, что от обычных криптографических хэш-функций универсальные функции отличаются более жесткими требованиями к частоте возникновения коллизий и прообразов II рода, чем указано в свойствах 2, 3 и 4. Усилить эти требования удается за счет того, что для хэширования используется не одна какая-либо фиксированная хэш-функция $H(M)$, а целое семейство $\Phi=\{H_1(M),H_2(M),...,H_m(M)\}$ равноценных хэш-функций, причем для каждого нового сообщения конкретная функция $H_i(M)\in\Phi$ выбирается наугад с равной вероятностью $1/m$. Такая организация процесса хэширо-

вания гарантирует криптостойкость даже в том случае, если криптоаналитик намеренно подменяет сообщения *M* заранее известными ему текстами. Общая идеология создания универсальных хэш-функций и доказательство основной леммы усиления секретности изложены в работах [17,18].

Мощь усиления секретности состоит в том, что эта процедура позволяет сколь угодно минимизировать эффект утечки любого количества битов исходного ключа *K*. Развитие аппарата универсальных хэш-функций является одной из приоритетных задач современной криптографии.

Описание алгоритма хэширования MD4

Одним из способов построения 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].

По современным представлениям, алгоритм MD4 считается нестойким. Тем не менее, он прекрасно подходит для учебных целей изучения свойств хэш-функций, так как, с одной стороны, имеет относительно простую программную реализацию, а с другой стороны – архитектурно содержит почти все структурные компоненты, присущие современным хэш-функциям.

Алгоритм MD4 формирует 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 символом *l*.

Словом будем называть 32-разрядное целое значение (unsigned int в языках C/C++). Слово рассматривается как группа из четырёх байтов, в которой младший байт является первым байтом тетрады (располагается в близлежащем конце памяти). Например, тетраде байтов (0x6F, 0x02, 0xE5, 0x8C) соответствует слово со значением 0x8CE5026F (префикс 0x обозначает запись числа в 16-ричной системе счисления). Обратный порядок байтов является естественным

для большинства типов ЭВМ, так как считывание двоичных данных начинается с младших байтов и постепенно движется к старшему байту. Поэтому, если рассматривать слово как массив из 4-х байтов, нулевой элемент этого массива будет равен 0x6F.

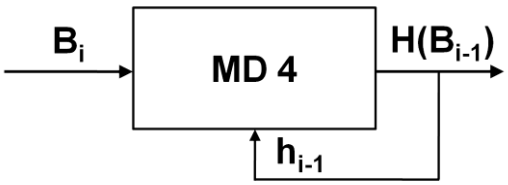

Рис. 4.2 – Принцип зависимого блочного хэширования сообщения.

Блоком будем называть последовательность из 512 битов (16 слов). Исходное сообщение разбивается на блоки B_i длиной 512 бит, каждый из которых последовательно подвергается процедуре хэширования. Результат хэширования предыдущего блока $h_{i-1} = H(B_{i-1})$ используется при вычислении хэша следующего блока. Хэш-кодом всего сообщения является результат обработки последнего блока. Перед хэшированием сообщение расширяется так, чтобы его 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 расширенного 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 сообщения 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]). Прежде чем подать блок на вход хэш-функции, порядок слов в нем обращается: $B_i = (X[15], X[14], ..., X[0])$. Далее в пределах каждого слова $X[j]$, $(j = \overline{0...15})$ обращается порядок байтов. Для каждого блока B_i выполняются 3 раунда преобразований. Введём следующие обозначения.

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

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

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 переносятся 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 побитовую инверсию (отрицание, 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 512-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 тестовых 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 $h \oplus h'$.

Построить график зависимости количества битов N_{av}, изменяющихся в хэш-коде при изменении k битов во входном сообщении. Объяснить полученный график. Каким должен быть этот график для идеальной хэш-функции?

Задание 3. Написать программу, реализующую процедуру поиска коллизий алгоритма MD4, то есть двух строк, имеющих одинаковый хэш-код. Для сокращения числа операций ($\sim 2^{n/2}$) 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 выполняется следующим образом. Последовательно генерируются псевдослучайные строки фиксированной длины L (задаётся пользователем) и вычисляются их хэш-коды. Для хранения результатов этих опера-

ций используются 2 массива: в один массив $\{M\}$ записываются сгенерированные строки, а во второй $\{h\}$ – соответствующие им хэш-коды. Указанные действия продолжать до обнаружения двух одинаковых элементов в массиве хэш-кодов $\{h\}$. Вывести на экран строки M' и M'' – обнаруженную коллизию, их общий хэш-код h, а также общее количество сгенерированных строк N. Построить график зависимости $N(k)$.

Задание 4. Программно реализовать поиск прообраза для заданного хэш-кода h некоторой строки M. В качестве хэш-кода также использовать только часть реального хэша (см. Задание 3). Исходным сообщением M является парольная фраза (вводится пользователем), для которой вычисляется частичный хэш-код длиной k младших битов слова A (k задаётся пользователем). Путем формирования псевдослучайных строк найти строку M', дающую тот же самый частичный хэш-код. Вывести найденную строку M' 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 $N(k)$.

Задание 5. Исследовать влияние инициализирующего состояния служебных регистров на качество хэширования. Для этого произвольным образом изменить инициализирующие константы A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476 в алгоритме хэширования MD4 и построить график зависимости $N_{av}(k)$ для лавинного эффекта. Сравнить графики до (см. задание 2) и после изменения инициализирующих констант. Объяснить наблюдающиеся закономерности.

Задание 6(*). Исследовать влияние количества раундов на качество хэширования. Для этого искусственным образом заблокировать в алгоритме хэширования MD4 выполнение третьего раунда и построить график зависимости $N_{av}(k)$ для лавинного эффекта. Сравнить графики для 3-раундового (полного) и 2-раундового (усеченного) хэширования. Аналогичным образом исследовать

качество перемешивания данных в 1-раундовом алгоритме MD4, заблокировав выполнение второго и третьего раундов. Объяснить наблюдающиеся закономерности.

Задание 7(*). Построить график зависимости длительности $t(V)$ хэширования от битовой длины V сообщения. Объяснить полученный график.

Задание 8(**). Исследовать возможность использования хэш-функции для генерации ключевой последовательности. Допустим, нам требуется создать ключевую последовательность битовой длины L. Известно, что при хэшировании одного сообщения алгоритм MD4 формирует 128-битный код h, близкий по своим свойствам к равномерной двоичной строке. Следовательно, для создания ключевой последовательности длиной L требуется обработать (L/128) сообщений и конкатенировать их хэш-коды. В качестве входных сообщений использовать псевдослучайные строки длиной k символов (вводится пользователем).

Протестировать полученную ключевую последовательность с помощью статистических тестов из Лабораторной работы №2. Для этого длина L (вводится пользователем) должна быть не менее 20000. Объяснить результаты тестирования. Влияет ли длина хэшируемых сообщений k на качество генерируемой ключевой последовательности?

Контрольные вопросы

1. Что такое хэш-функции? Для чего они применяются?
2. Что такое односторонность хэш-фукнции?
3. Что такое коллизия? Каковы причины её возникновения? Является ли она желательным или негативным эффектом?
4. Что такое прообраз второго рода? Какую угрозу безопасности он несет?
5. Что такое «лавинный эффект»? За счет чего он достигается?
6. За счет чего достигается возможность хэширования сообщений неограниченной длины?

7. В чем основное назначение раундовых операций алгоритма хэширования?
8. Как можно повысить криптостойкость хэширования?
9. Что такое усиление секретности?
10. Что такое ε-секретность ключа шифрования?
11. В чем сходство процедуры усиления секретности с экстракторами случайности?

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 решение типовых задач, встречающихся 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 независимо 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 отличен от нуля, 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* позиций. 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* позиций 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 операции 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, 528с.
2. Шеннон Р. Имитационное моделирование систем – искусство и наука: Перевод с англ. – М.: Мир, 1978. – 418с.
3. Петраков А.В. Основы практической защиты информации. – М.: Радио и связь, 2000. – 362с.
4. Романец Ю.В., Тимофеев П.А. Защиты информации в компьютерных системах и сетях. – М.: Радио и связь, 2000. – 320с.
5. Жук А. П. Защита информации: Учебное пособие / А.П. Жук, Е.П. Жук, О.М. Лепешкин, А.И. Тимошкин. - 2-е изд. - М.: ИЦ РИОР: НИЦ ИНФРА-М, 2015. - 392 с.: http://znanium.com/bookread.php?book=474838
6. Практическая криптография: Пособие / Масленников М.Е. - СПб:БХВ-Петербург, 2015. - 465 с. http://znanium.com/bookread2.php?book=944503
7. Введение в криптографию. Курс лекций / В.А. Романьков. — 2-е изд., испр. и доп. — М. : ФОРУМ : ИНФРА-М, 2017. — 240 с. — (Высшее образование).
8. Шеннон, К.Э. Работы по теории информации и кибернетике / К. Э. Шеннон. – М.: Изд-во иностранной литературы, 1963. – 827 с.
9. Яглом, И.М. Вероятность и информация / И.М. Яглом, А.М. Яглом. – М.: КомКнига, 2007. – 512 с.
10. Кнут, Д. Искусство программирования для ЭВМ. Получисленные алгоритмы / Дональд Кнут; под ред. Ю.В. Козаченко. – 3-е изд. – М.: Вильямс, 2000. – 828 с.: ил.
11. Rukhin, A. A statistical test suite for random and pseudorandom number generators for cryptographic applications / A. Rukhin, J. Soto [et al.]. – Special Publication 800-22 Rev.1a. – National Institute of Standards and Technology (NIST), 2010. – 131 p.

12. Ilyas, A. Statistical analysis of pseudorandom binary sequences generated by using tent map / A. Ilyas, A. Vlad, A. Luca // UPB Scientific Bulletin. Series A. – 2013. – vol.75. – iss.3. – pp. 113-122.
13. Chor B., Goldreich O. Unbaised bits from sources of weak randomness and probabilistic communication complexity // SIAM J. on Computing. – 1988.– vol. 17, no. 2. – pp. 230-261.
14. Trevisan L., Vadhan S. Extracting randomness from samplable distributions // Proceedings of 41st Annual Symposium on Foundations of Computer Science. – 2000. – pp. 32-42.
15. Bennet, C.H. Generalized privacy amplification / C.H. Bennet, G. Brassard, C. Crepeau, U.M. Maurer // IEEE Trans. on Inf. Theory. – 1995. – vol. 41. –iss. 6. – pp. 1915-1923.
16. Assche, G. Quantum cryptography and secret-key distillation: Cambridge university press, 2006. – 261 p.
17. Hastad, J. A pseudorandom generator from any one-way function / J. Hastad, R. Impagliazzo, L.A. Levin, M. Luby // SIAM Journal on Computing. – 1999. – vol. 28(4). – pp. 1364-1396.
18. Dodis, Y. Fuzzy extractors: how to generate strong keys from biometrics and other noisy data / Y. Dodis, R. Ostrovsky, L. Reyzin, A. Smith // SIAM J. on Computting. – 2008. – vol. 38(1). – pp. 97-139.
19. Maurer, U. Unbreakable keys from random noise / U. Maurer, R. Renner, S. Wolf // Security with Noisy Data. London: Springer, 2007.