

Федеральное агентство по образованию
Пермский Государственный Технический Университет
Кафедра Информационных Технологий и Автоматизированных Систем

Белковский С.В., Мосиенко А.Ю.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторным работам по дисциплине
«МЕТОДЫ И СРЕДСТВА ЗАЩИТЫ ИНФОРМАЦИИ»

Для студентов направления
«Информатика и вычислительная техника»
(654600, 552800)
дневной, заочной и дистанционной
форм обучения

Пермь 2007

УДК 681.3

Рецензент

канд. техн. наук, доцент кафедры АТПП
Пермского государственного технического университета
Ю.Н. Липин

Белковский С.В., Мосиенко А.Ю.

Методические указания к лабораторным работам по дисциплине «Методы и средства защиты информации»: Для студентов направления «Информатика и вычислительная техника» (654600, 552800) дневной, заочной и дистанционной форм обучения / ПГТУ. – Пермь: ЦНТИ, 2007. – 92 с.

Приводятся методические указания к 8 лабораторным работам по дисциплине «Методы и средства защиты информации» для студентов направления «Информатика и вычислительная техника» (654600, 552800) дневной, заочной и дистанционной форм обучения. Каждое лабораторное занятие включает в себя краткие теоретические сведения, иллюстрируемые решениями типовых задач, а также задания для самостоятельного решения.

Утверждено на заседании кафедры ИТАС ПГТУ «___» _____ 2007 г.
протокол № ____.

© Пермский Государственный Технический Университет

Содержание

Введение.....	3
Лабораторная работа №1. Шифры перестановки и замены	4
Лабораторная работа №2. Алгоритм RSA	29
Лабораторная работа №3. Несимметричные алгоритмы шифрования	34
Лабораторная работа №4. Блочные шифры	40
Лабораторная работа №5. Шифры обнаружения и коррекции ошибок	53
Лабораторная работа №6. Методы сжатия информации	73
Лабораторная работа №7. Алгоритмы хеширования паролей	82
Лабораторная работа №8. Методы гаммирования	87
Список литературы	92

Введение

Активное развитие информационных технологий сегодня делает информацию и процесс ее получения намного более доступными. Интернет, спутниковая связь позволяют получать информацию в любой точке земли. Соответственно все более актуальной становится задача обеспечения сохранности информации, закрытой для посторонних лиц.

Если раньше вопросы обеспечения информационной безопасности касались, в основном, государственных служб, то сейчас эта область все больше внедряется в сферу экономики. Быстрое развитие научно-технического прогресса в двадцатом веке позволило интенсифицировать практически все процессы производства и внедрить новые технологии. При этом для достижения конкурентного преимущества применялись и применяются методы промышленного шпионажа и информационной разведки, с целью добывания сведений о достижениях и способе ведения производства конкурентов.

При этом уже становится недостаточным применение простых мер. Необходимо активное противодействие процессу хищения информации. Для ведения информационной разведки применяются все более изощренные программно-аппаратные технические средства. Квалификация информационных шпионов находится на очень высоком уровне.

Одними из методов обеспечения информационной безопасности являются методы шифрования, рассмотренные в данном методическом пособии. Они позволяют достаточно простыми средствами обеспечить приемлемый уровень конфиденциальности информации.

Шифры перестановки и замены относятся к самым простым и наименее криптостойким. Однако позволяют понять базовые принципы шифрования. Несимметричные алгоритмы шифрования, блочные шифры и методы гаммирования обладают намного большей криптостойкостью и повышенной сложностью реализации. Именно они находят сейчас самое широкое применение. Алгоритмы открытого ключа, такие как RSA, а также методы сжатия информации и хеширования паролей используются при передаче данных по информационным сетям, в том числе глобальной сети Internet.

Данное учебное пособие призвано повысить квалификацию в вопросах обеспечения информационной безопасности и может быть полезно не только студентам, а также работникам сферы информационных технологий.

Лабораторная работа №1. Шифры перестановки и замены
Цель работы: Получить практические навыки по применению шифров перестановки и шифров простой замены.

Теоретические сведения

Шифры перестановки

Шифр, преобразования из которого изменяют только порядок следования символов исходного текста, но не изменяют их самих, называется шифром перестановки (ШП).

Шифрами перестановки называются такие шифры, преобразования из которых приводят к изменению только порядка следования символов исходного сообщения. Примером преобразования, которое может содержаться в шифре перестановки, является следующее правило. Каждая буква исходного сообщения, стоящая в тексте на позиции с четным номером, меняется местами с предшествующей ей буквой. В этом случае ясно, что и исходное, и шифрованное сообщение состоят из одних и тех же букв.

Рассмотрим преобразование из ШП, предназначенное для шифрования сообщения длиной n символов. Его можно представить с помощью таблицы

$$\begin{pmatrix} 1 & 2 & \dots & n \\ i_1 & i_2 & \dots & i_n \end{pmatrix}$$

где i_1 - номер места шифртекста, на которое попадает первая буква исходного сообщения при выбранном преобразовании, i_2 - номер места для второй буквы и т.д. В верхней строке таблицы выписаны по порядку числа от 1 до n , а в нижней - те же числа, но в произвольном порядке. Такая таблица называется подстановкой степени n .

Зная подстановку, задающую преобразование, можно осуществить как шифрование, так и дешифрование текста. Например, если для преобразования используется подстановка

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 2 & 3 & 5 & 1 & 6 \end{pmatrix}$$

и в соответствии с ней зашифровывается слово МОСКВА, то получится КОСВМА. Попробуйте расшифровать сообщение НЧЕИУК, полученное в результате преобразования с помощью указанной выше подстановки.

В соответствии с методом математической индукции, можно легко убедиться в том, что существует $(1 \cdot 2 \cdot 3 \cdot \dots \cdot n)$ вариантов заполнения нижней строки таблицы (1). Таким образом, число различных преобразований шифра перестановки, предназначенного для шифрования сообщений длины n , меньше либо равно $n!$ (заметим, что в это число входит и вариант преобразования, оставляющий все символы на своих местах!).

С увеличением числа n значение $n!$ растет очень быстро. Приведем таблицу значений $n!$ для первых 10 натуральных чисел:

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

При больших n для приближенного вычисления $n!$ можно пользоваться известной формулой Стирлинга

$$n! \approx \sqrt{2 \cdot \pi \cdot n} \left(\frac{n}{e} \right)^n$$

где $e = 2,718281828\dots$

Примером ШП, предназначенного для шифрования сообщений длины n , является шифр, в котором в качестве множества ключей взято множество всех подстановок степени n , а соответствующие им преобразования шифра задаются, как было описано выше. Число ключей такого шифра равно $n!$.

Для использования на практике такой шифр не удобен, так как при больших значениях n приходится работать с длинными таблицами.

Широкое распространение получили шифры перестановки, использующие некоторую геометрическую фигуру. Преобразования из этого шифра состоят в том, что в фигуру исходный текст вписывается по ходу одного «маршрута», а затем по ходу другого выписывается с нее. Такой шифр называют маршрутной перестановкой. Например, можно вписывать исходное сообщение в прямоугольную таблицу, выбрав такой маршрут: по горизонтали, начиная с левого верхнего угла поочередно слева направо и справа налево. Выписывать же сообщение будем по другому маршруту: по вертикали, начиная с верхнего правого угла и двигаясь поочередно сверху вниз и снизу вверх.

Зашифруем, например, указанным способом фразу:

ПРИМЕРМАРШРУТНОЙПЕРЕСТАНОВКИ

используя прямоугольник размера 4 x 7:

П	Р	И	М	Е	Р	М
Н	Т	У	Р	Ш	Р	А
О	Й	П	Е	Р	Е	С
И	К	В	О	Н	А	Т

Зашифрованная фраза выглядит так:

МАСТАЕРРЕШРНОЕРМИУПВКЙТРПНОИ

Теоретически маршруты могут быть значительно более изощренными, однако запутанность маршрутов усложняет использование таких шифров.

Ниже приводятся описания трех разновидностей шифров перестановки:

Шифр «Сцитала». Одним из самых первых шифровальных приспособлений был жезл («Сцитала»), применявшийся еще во времена войны Спарты против Афин в V веке до н. э. Это был цилиндр, на который виток к витку наматывалась узкая папирусная лента (без просветов и нахлестов), а затем на этой ленте вдоль его оси записывался необходимый для передачи текст. Лента сматывалась с цилиндра и отправлялась адресату, который, имея цилиндр точно такого же диаметра, наматывал ленту на него и прочитывал сообщение. Ясно, что такой способ шифрования осуществляет перестановку местами букв сообщения.

Шифр «Сцитала» реализует не более n перестановок (n , по-прежнему, - длина сообщения). Действительно, этот шифр, как нетрудно видеть, эквивалентен следующему шифру маршрутной перестановки: в таблицу, состоящую из m столбцов, построчно записывают сообщение, после чего выписывают буквы по столбцам. Число задействованных столбцов таблицы не может превосходить длины сообщения.

Имеются еще и чисто физические ограничения, накладываемые реализацией шифра «Сцитала». Естественно предположить, что диаметр жезла не должен превосходить 10 сантиметров. При высоте строки в 1 сантиметр на одном витке такого жезла уместится не более 32 букв ($10\pi < 32$). Таким образом, число перестановок, реализуемых «Сциталой», вряд ли превосходит 32.

Шифр «Поворотная решетка». Для использования шифра, называемого поворотной решеткой, изготавливается трафарет из прямоугольного листа клетчатой бумаги размера $2m \times 2k$ клеток. В трафарете вырезано mk клеток так, что при наложении его на чистый лист бумаги того же размера четырьмя возможными способами его вырезы полностью покрывают всю площадь листа.

Буквы сообщения последовательно вписываются в вырезы трафарета (по строкам, в каждой строке слева направо) при каждом из четырех его возможных положений в заранее установленном порядке.

Поясним процесс шифрования на примере. Пусть в качестве ключа используется решетка 6×10 , приведенная на рис. 1.

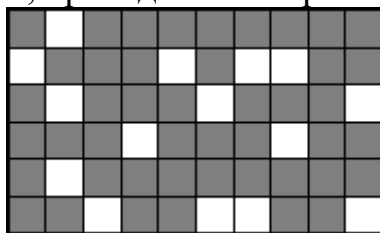


Рис. 1. Ключ шифрования для метода «Поворотной решетки».

Зашифруем с ее помощью текст
**ШИФРРЕШЕТКАЯВЛЯЕТСЯЧАСТНЫМСЛУЧАЕМШИФРА-
МАРШРУТНОЙПЕРЕСТАНОВКИ**

Наложив решетку на лист бумаги, вписываем первые 15 (по числу вырезов) букв сообщения: ШИФРРЕШЕТКАЯВЛЯ.... Сняв решетку, мы увидим текст, представленный на рис. 2. Поворачиваем решетку на 180°. В окошечках появятся новые, еще не заполненные клетки. Вписываем в них следующие 15 букв. Получится запись, приведенная на рис. 3. Затем переворачиваем решетку на другую сторону и зашифровываем остаток текста аналогичным образом (рис. 4, 5).

	Ш							
И				Ф		Р	Р	
	Е				Ш			Е
			Т				К	
	А							
		Я			В	Л		Я

Рис. 2.

Е	Ш		Т	С			Я	
И				Ф		Р	Р	Ч
	Е	А			Ш	С		Е
Т			Т	Н			К	Ы
	А	М	С		Л			У
		Я			В	Л		Ч

Рис. 3.

Е	Ш	А	Т	С	Е	М	Я		Ш
И	И			Ф		Р	Р	Ч	
	Е	А	Ф		Ш	С	Р		Е
Т	А		Т	Н	М		К	Ы	А
Р	А	М	С	Ш	Л	Р	У		У
	Т	Я			В	Л		Ч	Я

Рис. 4.

Е	Ш	А	Т	С	Е	М	Я	Н	Ш
И	И	О	Й	Ф	П	Р	Р	Ч	Е
Р	Е	А	Ф	Е	Ш	С	Р	С	Е
Т	А	Т	Т	Н	М	А	К	Ы	А
Р	А	М	С	Ш	Л	Р	У	Н	У
О	Т	Я	В	К	В	Л	И	Ч	Я

Рис. 5.

Получатель сообщения, имеющий точно такую же решетку, без труда прочтет исходный текст, наложив решетку на шифртекст по порядку четырьмя способами.

Можно доказать, что число возможных трафаретов, то есть количество ключей шифра «решетка», составляет $T = 4^{mk}$. Этот шифр предназначен для сообщений длины $n = 4^{mk}$. Число всех перестановок в тексте такой длины составит $(4mk)!$, что во много раз больше числа T . Однако, уже при

размере трафарета 8x8 число возможных решеток превосходит 4 миллиарда.

Широко распространена разновидность шифра маршрутной перестановки, называемая «**шифром вертикальной перестановки**» (ШВП). В нем снова используется прямоугольник, в который сообщение вписывается обычным способом (по строкам слева направо). Выписываются буквы по вертикали, а столбцы при этом берутся в порядке, определяемом ключом. Пусть, например, этот ключ таков: (5,4,1,7,2,6,3), и с его помощью надо зашифровать сообщение:

ВОТПРИМЕРШИФРАВЕРТИКАЛЬНОЙПЕРЕСТАНОВКИ

Впишем сообщение в прямоугольник, столбцы которого пронумерованы в соответствии с ключом:

5	1	4	7	2	6	3
В	О	Т	П	Р	И	М
Е	Р	Ш	И	Ф	Р	А
В	Е	Р	Т	И	К	А
Л	Ь	Н	О	Й	П	Е
Р	Е	С	Т	А	Н	О
В	К	И	-	-	-	-

Теперь, выбирая столбцы в порядке, заданном ключом, и выписывая последовательно буквы каждого из них сверху вниз, получаем такую криптограмму:

ОРЕЬЕКРФИЙА-МАОЕО-ТШРНСИВЕВЛРВИРКПН-ПИТОТ-

Число ключей ШВП не более $m!$, где m - число столбцов таблицы. Как правило, m гораздо меньше, чем длина текста n (сообщение укладывается в несколько строк по m букв), а, значит, и $m!$ много меньше $n!$.

Пользуясь приведенной выше **формулой Стирлинга** при больших m и n , попытайтесь оценить, во сколько раз число возможных перестановок ШВП с m столбцами меньше числа всех перестановок на тексте длины n , кратном m .

В случае, когда ключ ШВП не рекомендуется записывать, его можно извлекать из какого-то легко запоминающегося слова или предложения. Для этого существует много способов. Наиболее распространенный состоит в том, чтобы приписывать буквам числа в соответствии с обычным алфавитным порядком букв. Например, пусть ключевым словом будет ПЕРЕСТАНОВКА. Присутствующая в нем буква А получает номер 1. Если какая-то буква входит несколько раз, то ее появления нумеруются после-

довательно слева направо. Поэтому второе вхождение буквы А получает номер 2. Поскольку буквы Б в этом слове нет, то буква В получает номер 3 и так далее. Процесс продолжается до тех пор, пока все буквы не получат номера. Таким образом, мы получаем следующий ключ:

П	Е	Р	Е	С	Т	А	Н	О	В	К	А
9	4	10	5	11	12	1	7	8	3	6	2

Шифры простой замены

При шифровании заменой (подстановкой) символы шифруемого текста заменяются символами того же или другого алфавита с заранее установленным правилом замены. В шифре простой замены каждый символ исходного текста заменяется символами того же алфавита одинаково на всем протяжении текста. Часто шифры простой замены называют шифрами одноалфавитной подстановки.

Полибианский квадрат

Одним из первых шифров простой замены считается так называемый *полибианский квадрат*. За два века до нашей эры греческий писатель и историк Полибий изобрел для целей шифрования квадратную таблицу размером 5x5, заполненную буквами греческого алфавита в случайном порядке.

q	p	w	o	e
i	r	u	t	y
z	m	x	n	c
b	v	a	l	s
k	d	j	f	h

Полибианский квадрат, заполненный случайным образом буквами латинского алфавита и пробелом

При шифровании в этом полибианском квадрате находили очередную букву открытого текста и записывали в шифртекст букву, расположенную ниже ее в том же столбце. Если буква текста оказывалась в нижней строке таблицы, то для шифртекста брали самую верхнюю букву из того же столбца. Например, для слова

square

получается шифртекст

hixjmy

Концепция полибианского квадрата оказалась плодотворной и нашла применение в криптосистемах последующего времени.

Система шифрования Цезаря

Шифр Цезаря является частным случаем шифра простой замены (одноалфавитной подстановки). Свое название этот шифр получил по имени римского императора Гая Юлия Цезаря, который использовал этот шифр при переписке с Цицероном (около 50 г. до н.э.).

При шифровании исходного текста каждая буква заменялась на другую букву того же алфавита по следующему правилу. Заменяющая буква определялась путем смещения по алфавиту от исходной буквы на K букв. При достижении конца алфавита выполнялся циклический переход к его началу. Цезарь использовал шифр замены при смещении $K = 3$. Такой шифр замены можно задать таблицей подстановок, содержащей соответствующие пары букв открытого текста и шифртекста. Совокупность возможных подстановок для $K = 3$ показана в следующей таблице.

A -> D	J -> M	S -> V
B -> E	K -> N	T -> W
C -> F	L -> O	U -> X
D -> G	M -> P	V -> Y
E -> H	N -> Q	W -> Z
F -> I	O -> R	X -> A
G -> J	P -> S	Y -> B
H -> K	Q -> T	Z -> C
I -> L	R -> U	

Например, послание Цезаря

VENI VIDI VICI

в переводе на русский означает "Пришел, Увидел, Победил"), направленное его другу Аминтию после победы над понтийским царем Фарнаком, сыном Митридата, выглядело бы в зашифрованном виде так:

YHQL YLGL YLFL

Концепция, заложенная в систему шифрования Цезаря, оказалась весьма плодотворной, о чем свидетельствуют ее многочисленные модификации. Несколько таких модификаций будут рассмотрены ниже.

Аффинная система подстановок Цезаря

В данном преобразовании буква, соответствующая числу t , заменяется на букву, соответствующую числовому значению $(at+b)$ по модулю m .

Следует заметить, что a и m должны быть взаимно простыми числами.

Например, пусть $m=26$, $a=3$, $b=5$. Тогда, очевидно, НОД $(3,26)=1$, и мы получаем следующее соответствие между числовыми кодами букв:

t	0	1	2	3	4	...	23	24	25
3t+5	5	8	11	14	17	...	22	25	2

Преобразуя числа в буквы английского языка, получаем следующее соответствие для букв открытого текста и шифртекста:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	I	L	O	R	U	X	A	D	G	J	M	P	S	V	Y	B	E	H	K	N	Q	T	W	Z	C

Исходное сообщение NOPE преобразуется в шифртекст AVYR

Достоинством аффинной системы является удобное управление ключами - ключи шифрования и дешифрования представляются в компактной форме в виде пары чисел (a, b). Недостатки аффинной системы аналогичны недостаткам системы шифрования Цезаря.

Аффинная система использовалась на практике несколько веков назад, а сегодня ее применение ограничивается большей частью иллюстрациями основных криптологических положений.

Система Цезаря с ключевым словом

Система шифрования Цезаря с ключевым словом является одноалфавитной системой подстановки. Особенностью этой системы является использование ключевого слова для смещения и изменения порядка символов в алфавите подстановки.

Выберем некоторое число k , $0 \leq k \leq 25$, и слово или короткую фразу в качестве *ключевого слова*. Желательно, чтобы все буквы ключевого слова были различными. Пусть выбраны слово DIPLOMAT в качестве ключевого слова и число $k = 5$.

Ключевое слово записывается под буквами алфавита, начиная с буквы, числовой код которой совпадает с выбранным числом k :

0	1	2	3	4	5					10					15					20					25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
					D	I	P	L	O	M	A	T													

Оставшиеся буквы алфавита подстановки записываются после ключевого слова в алфавитном порядке:

					5																				
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
V	W	X	Y	Z	D	I	P	L	O	M	A	T	B	C	E	F	G	H	J	K	N	Q	R	S	U

Теперь мы имеем подстановку для каждой буквы произвольного сообщения.

Исходное сообщение HELLO шифруется как PZAAC

Несомненным достоинством системы Цезаря с ключевым словом является то, что количество возможных ключевых слов практически неисчерпаемо. Недостатком этой системы является возможность взлома шифртекста на основе анализа частот появления букв.

Шифрующие таблицы Трисемуса

В 1508 г. аббат из Германии Иоганн Трисемус написал печатную работу по криптологии под названием «Полиграфия». В этой книге он впервые систематически описал применение шифрующих таблиц, заполненных алфавитом в случайном порядке. Для получения такого шифра замены обычно использовались таблица для записи букв алфавита и ключевое слово (или фраза). В таблицу сначала вписывалось по строкам ключевое слово, причем повторяющиеся буквы отбрасывались. Затем эта таблица дополнялась не вошедшими в нее буквами алфавита по порядку.

Поскольку ключевое слово или фразу легко хранить в памяти, то такой подход упрощал процессы шифрования и дешифрования. Поясним этот метод шифрования на примере. Для русского алфавита шифрующая таблица может иметь размер 4x8. Выберем в качестве ключа слово БАНДЕРОЛЬ.

Б	А	Н	Д	Е	Р	О	Л
Ь	В	Г	Ж	З	И	Й	К
М	П	С	Т	У	Ф	Х	Ц
Ч	Ш	Щ	Ы	Ъ	Э	Ю	Я

Как и в случае полибианского квадрата, при шифровании находят в этой таблице очередную букву открытого текста и записывают в шифртекст букву, расположенную ниже ее в том же столбце. Если буква текста оказывается в нижней строке таблицы, тогда для шифртекста берут самую верхнюю букву из того же столбца.

Например, при шифровании с помощью этой таблицы сообщения

ВЫЛЕТАЕМ ПЯТОГО

получаем шифртекст

ПДКЗЫВЗЧШЛЫЙСЙ

Такие табличные шифры называются монограммными, так как шифрование выполняется по одной букве. Трисемус первым заметил, что шифрующие таблицы позволяют шифровать сразу по две буквы. Такие шифры называются биграммными.

Биграммный шифр Плейфейра

Шифр Плейфейра, изобретенный в 1854 г., является наиболее известным биграммным шифром замены. Он применялся Великобританией во время первой мировой войны. Основой шифра Плейфейра является шифрующая таблица со случайно расположенными буквами алфавита исходных сообщений.

Для удобства запоминания шифрующей таблицы отправителем и получателем сообщений можно использовать ключевое слово (или фразу) при заполнении начальных строк таблицы. В целом структура шифрующей таблицы системы Плейфейра полностью аналогична структуре шифрующей таблицы Трисемуса. Поэтому для пояснения процедур шифрования и расшифрования в системе Плейфейра воспользуемся шифрующей таблицей Трисемуса из предыдущего раздела.

Процедура шифрования включает следующие шаги:

1. Открытый текст исходного сообщения разбивается на пары букв (биграммы). Текст должен иметь четное количество букв и в нем не должно быть биграмм, содержащих две одинаковые буквы. Если эти требования не выполнены, то текст модифицируется даже из-за незначительных орфографических ошибок.

2. Последовательность биграмм открытого текста преобразуется с помощью шифрующей таблицы в последовательность биграмм шифртекста по следующим правилам:

- 2а. Если обе буквы биграммы открытого текста не попадают на одну строку или столбец (как, например, буквы А и Й), тогда находят буквы в углах прямоугольника, определяемого данной парой букв. (В нашем примере это буквы АЙОВ. Пара букв АЙ отображается в пару ОВ. Последовательность букв в биграмме шифртекста должна быть зеркально расположенной по отношению к последовательности букв в биграмме открытого текста).

- 2б. Если обе буквы биграммы открытого текста принадлежат одному столбцу таблицы, то буквами шифртекста считаются буквы, которые лежат под ними. (Например, биграмма НС дает биграмму шифртекста ГЩ.) Если при этом буква открытого текста находится в нижней строке, то для шифртекста берется соответствующая буква из верхней строки того же столбца. (Например, биграмма ВШ дает биграмму шифртекста ПА.)

- 2в. Если обе буквы биграммы открытого текста принадлежат одной строке таблицы, то буквами шифртекста считаются буквы, ко-

торые лежат справа от них. (Например, биграмма НО дает биграмму шифртекста ДЛ.) Если при этом буква открытого текста находится в крайнем правом столбце, то для шифра берут соответствующую букву из левого столбца в той же строке. (Например, биграмма ФЦ дает биграмму шифртекста ХМ.).

Зашифруем текст

ВСЕ ТАЙНОЕ СТАНЕТ ЯВНЫМ

Разбиение этого текста на биграммы дает

ВС ЕТ АЙ НО ЕС ТА НЕ ТЯ ВН ЫМ

Данная последовательность биграмм открытого текста преобразуется с помощью шифрующей таблицы в следующую последовательность биграмм шифртекста

ГП ДУ ОВ ДЛ НУ ПД ДР ЦЫ ГА ЧТ

При дешифровании применяется обратный порядок действий.

Следует отметить, что шифрование биграммами резко повышает стойкость шифров к вскрытию. Хотя книга И.Трисемуса «Полиграфия» была относительно доступной, описанные в ней идеи получили признание лишь спустя три столетия. По всей вероятности, это было обусловлено плохой осведомленностью криптографов о работах богослова и библиофила Трисемуса в области криптографии.

Система омофонов

Система омофонов обеспечивает простейшую защиту от криптоаналитических атак, основанных на подсчете частот появления букв в шифртексте. Система омофонов является одноалфавитной, хотя при этом буквы исходного сообщения имеют несколько замен. Число замен берется пропорциональным вероятности появления буквы в открытом тексте.

Данные о распределениях вероятностей букв в русском и английском текстах приведены в таблицах. Буквы в таблицах указаны в порядке убывания вероятности их появления в тексте. Например, русская буква Е встречается в 36 раз чаще, чем буква Ф, а английская буква Е встречается в 123 раза чаще, чем буква Z.

Шифруя букву исходного сообщения, выбирают случайным образом одну из ее замен. Замены (часто называемые омофонами) могут быть представлены трехразрядными числами от 000 до 999. Например, в английском алфавите букве Е присваиваются 123 случайных номера, буквам В и G - по

16 номеров, а буквам J и Z - по 1 номеру. Если омофоны (замены) присваиваются случайным образом различным появлениям одной и той же буквы, тогда каждый омофон появляется в шифртексте равновероятно.

Распределение вероятностей букв в русских текстах

Буква	Вероятность	Буква	Вероятность	Буква	Вероятность	Буква	Вероятность
Пробел	0,175	Р	0,040	Я	0,018	Х	0,009
О	0,090	В	0,038	Ы	0,016	Ж	0,007
Е	0,072	Л	0,035	З	0,016	Ю	0,006
А	0,062	К	0,028	Ь	0,014	Ш	0,006
И	0,062	М	0,026	Б	0,014	Ц	0,004
Н	0,053	Д	0,025	Г	0,013	Щ	0,003
Т	0,053	П	0,023	Ч	0,012	Э	0,003
С	0,045	У	0,021	Й	0,010	Ф	0,002

Распределение вероятностей букв в английских текстах

Буква	Вероятность	Буква	Вероятность	Буква	Вероятность
Е	0,123	Л	0,040	В	0,016
Т	0,096	Д	0,036	Г	0,016
А	0,081	С	0,032	У	0,009
О	0,079	И	0,031	К	0,005
Н	0,072	Р	0,023	Q	0,002
И	0,071	Ф	0,023	Х	0,002
S	0,066	М	0,022	Ј	0,001
В	0,060	W	0,020	Z	0,001
Н	0,051	У	0,019		

При таком подходе к формированию шифртекста простой подсчет частот уже ничего не дает криптоаналитику. Однако в принципе полезна также информация о распределении пар и троек букв в различных естественных языках. Если эту информацию использовать при криптоанализе, он будет проведен более успешно.

Шифры сложной замены

Шифры сложной замены называют многоалфавитными, так как для шифрования каждого символа исходного сообщения применяют свой шифр простой замены. Многоалфавитная подстановка последовательно и циклически меняет используемые алфавиты.

При r -алфавитной подстановке символ x_0 исходного сообщения заменяется символом y_0 из алфавита B_0 , символ x_1 -символом y_1 из алфавита B_1 , и так далее, символ x_{r-1} заменяется символом y_{r-1} из алфавита B_{r-1} , символ x_r заменяется символом y_r снова из алфавита B_0 , и т.д.

Общая схема многоалфавитной подстановки для случая $r = 4$ приведена ниже.

Входной символ:	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
Алфавит подстановки:	B_0	B_1	B_2	B_3	B_0	B_1	B_2	B_3	B_0	B_1

Эффект использования многоалфавитной подстановки заключается в том, что обеспечивается маскировка естественной статистики исходного языка, так как конкретный символ из исходного алфавита A может быть преобразован в несколько различных символов шифровальных алфавитов B_j . Степень обеспечиваемой защиты теоретически пропорциональна длине периода r в последовательности используемых алфавитов B_j .

Многоалфавитные шифры замены предложил и ввел в практику криптографии Леон Батист Альберти, который также был известным архитектором и теоретиком искусства. Его книга «Трактат о шифре», написанная в 1566 г., представляла собой первый в Европе научный труд по криптологии. Кроме шифра многоалфавитной замены, Альберти также подробно описал устройства из вращающихся колес для его реализации. Криптологи всего мира почитают Л.Альберти основоположником криптологии.

Шифр Гронсфельда

Этот шифр сложной замены, называемый шифром Гронсфельда, представляет собой модификацию шифра Цезаря числовым ключом. Для этого под буквами исходного сообщения записывают цифры числового ключа. Если ключ короче сообщения, то его запись циклически повторяют. Шифртекст получают примерно, как в шифре Цезаря, но отсчитывают по алфавиту не третью букву (как это делается в шифре Цезаря), а выбирают ту букву, которая смещена по алфавиту на соответствующую цифру ключа. Например, применяя в качестве ключа группу из четырех начальных цифр числа e (основания натуральных логарифмов), а именно 2718, получаем для исходного сообщения ВОСТОЧНЫЙ ЭКСПРЕСС следующий шифртекст:

Сообщение		В	О	С	Т	О	Ч	Н	Ы	Й		Э	К	С	П	Р	Е	С	С
Ключ		2	7	1	8	2	7	1	8	2		7	1	8	2	7	1	8	2
Шифртекст		Д	Х	Т	Ь	Р	Ю	О	Г	Л		Д	Л	Щ	С	Ч	Ж	Щ	У

Чтобы зашифровать первую букву сообщения В, используя первую цифру ключа 2, нужно отсчитать вторую по порядку букву от В в алфавите

В	Г	Д
	1	2

получается первая буква шифр-текста Д.

Следует отметить, что шифр Гронсфельда вскрывается относительно легко, если учесть, что в числовом ключе каждая цифра имеет только десять значений, а значит, имеется лишь десять вариантов прочтения каждой буквы шифртекста. С другой стороны, шифр Гронсфельда допускает дальнейшие модификации, улучшающие его стойкость, в частности двойное шифрование разными числовыми ключами.

Шифр Гронсфельда представляет собой по существу частный случай системы шифрования Вижинера.

Система шифрования Вижинера

Система Вижинера впервые была опубликована в 1586 г. и является одной из старейших и наиболее известных многоалфавитных систем. Свое название она получила по имени французского дипломата XVI века Блеза Вижинера, который развивал и совершенствовал криптографические системы.

Система Вижинера подобна такой системе шифрования Цезаря, у которой ключ подстановки меняется от буквы к букве. Этот шифр многоалфавитной замены можно описать таблицей шифрования, называемой таблицей (квадратом) Вижинера. Ниже показаны таблицы Вижинера для русского и английского алфавитов соответственно.

Таблица Вижинера используется для шифрования и дешифрования. Таблица имеет два входа:

- верхнюю строку подчеркнутых символов, используемую для считывания очередной буквы исходного открытого текста;
- крайний левый столбец ключа.

Последовательность ключей обычно получают из числовых значений букв ключевого слова.

При шифровании исходного сообщения его выписывают в строку, а под ним записывают ключевое слово (или фразу). Если ключ оказался короче сообщения, то его циклически повторяют. В процессе шифрования находят в верхней строке таблицы очередную букву исходного текста и в левом столбце очередное значение ключа. Очередная буква шифртекста находится на пересечении столбца, определяемого шифруемой буквой, и строки, определяемой числовым значением ключа.

Ключ	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э
0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э
1	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю
2	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
3	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а
4	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б
5	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в
6	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г
7	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д
8	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е
9	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж
10	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з
11	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и
12	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й
13	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к
14	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л
15	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м
16	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н
17	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о
18	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
19	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р
20	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с
21	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т
22	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у
23	ч	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф
24	ш	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х
25	щ	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц
26	ъ	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч
27	ы	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш
28	ь	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ
29	э	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ
30	ю	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы
31	я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь

Таблица Вижинера для русского алфавита

Ключ	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Таблица Вижинера для английского алфавита

Рассмотрим пример получения шифртекста с помощью таблицы Вижинера. Пусть выбрано ключевое слово АМБРОЗИЯ. Необходимо зашифровать сообщение ПРИЛЕТАЮ СЕДЬМОГО.

Выпишем исходное сообщение в строку и запишем под ним ключевое слово с повторением. В третью строку будем выписывать буквы шифртекста, определяемые из таблицы Вижинера.

Сообщение	П	Р	И	Л	Е	Т	А	Ю		С	Е	Д	Ь	М	О	Г	О
Ключ	А	М	Б	Р	О	З	И	Я		А	М	Б	Р	О	З	И	Я
Шифртекст	П	Ъ	Й	Ы	У	Щ	И	Э		С	С	Е	К	Ь	Х	Л	Н

Шифр «двойной квадрат» Уитстона

В 1854 г. англичанин Чарльз Уитстон разработал новый метод шифрования биграммами, который называют «двойным квадратом». Свое название этот шифр получил по аналогии с по-либианским квадратом. Шифр Уитстона открыл новый этап в истории развития криптографии. В отличие от полибианского шифр «двойной квадрат» использует сразу две таблицы, размещенные по одной горизонтали, а шифрование идет биграммами, как в шифре Плейфейра. Эти не столь сложные модификации привели к появлению на свет качественно новой криптографической системы ручного шифрования. Шифр «двойной квадрат» оказался очень надежным и удобным и применялся Германией даже в годы второй мировой войны.

Поясним процедуру шифрования этим шифром на примере. Пусть имеются две таблицы со случайно расположенными в них русскими алфавитами. Перед шифрованием исходное сообщение разбивают на биграммы. Каждая биграмма шифруется отдельно. Первую букву биграммы находят в левой таблице, а вторую букву - в правой таблице. Затем мысленно строят прямоугольник так, чтобы буквы биграммы лежали в его противоположных вершинах. Другие две вершины этого прямоугольника дают буквы биграммы шифртекста.

Ж	Щ	Н	Ю	Р
И	Т	Ь	Ц	Б
Я	М	Е	.	С
В	Ы	П	Ч	
:	Д	У	О	К
З	Э	Ф	Г	Ш
Х	А	,	Л	Ъ

И	Ч	Г	Я	Т
,	Ж	Ь	М	О
З	Ю	Р	В	Щ
Ц	:	П	Е	Л
Ъ	А	Н	.	Х
Э	К	С	Ш	Д
Б	Ф	У	Ы	

Две таблицы со случайно расположенными символами русского алфавита для шифра «двойной квадрат»

Предположим, что шифруется биграмма исходного текста ИЛ. Буква И находится в столбце 1 и строке 2 левой таблицы. Буква Л находится в столбце 5 и строке 4 правой таблицы. Это означает, что прямоугольник образован строками 2 и 4, а также столбцами 1 левой таблицы и 5 правой таблицы. Следовательно, в биграмму шифртекста входят буква О, расположенная в столбце 5 и строке 2 правой таблицы, и буква В, расположенная в столбце 1 и строке 4 левой таблицы, т.е. получаем биграмму шифртекста ОВ.

Если обе буквы биграммы сообщения лежат в одной строке, то и буквы шифртекста берут из этой же строки. Первую букву биграммы шифртекста берут из левой таблицы в столбце, соответствующем второй букве биграммы сообщения. Вторая же буква биграммы шифртекста берется из правой таблицы в столбце, соответствующем первой букве биграммы сообщения. Поэтому биграмма сообщения ТО превращается в биграмму шифртекста ЖБ. Аналогичным образом шифруются все биграммы сообщения:

Сообщение ПР ИЛ ЕТ АЮ _Ш ЕС ТО ГО

Шифртекст ПЕ ОВ ЦН ФМ ЕШ РФ БЖ ДЦ

Шифрование методом «двойного квадрата» дает весьма устойчивый к вскрытию и простой в применении шифр. Взламывание шифртекста «двойного квадрата» требует больших усилий, при этом длина сообщения должна быть не менее тридцати строк.

Одноразовая система шифрования

Почти все применяемые на практике шифры характеризуются как условно надежные, поскольку они могут быть в принципе раскрыты при

наличии неограниченных вычислительных возможностей. Абсолютно надежные шифры нельзя разрушить даже при использовании неограниченных вычислительных возможностей. Существует единственный такой шифр, применяемый на практике, - одноразовая система шифрования. Характерной особенностью одноразовой системы шифрования является одноразовое использование ключевой последовательности.

Одноразовая система изобретена в 1917 г. американцами Дж.Моборном и Г.Вернамом. Для реализации этой системы подстановки иногда используют одноразовый блокнот. Этот блокнот составлен из отрывных страниц, на каждой из которых напечатана таблица со случайными числами (ключами) K_i . Блокнот выполняется в двух экземплярах: один используется отправителем, а другой - получателем. Для каждого символа X_i сообщения используется свой ключ K_i из таблицы только один раз. После того как таблица использована, она должна быть удалена из блокнота и уничтожена. Шифрование нового сообщения начинается с новой страницы.

Этот шифр абсолютно надежен, если набор ключей K_i действительно случаен и непредсказуем. Если криптоаналитик попытается использовать для заданного шифртекста все возможные наборы ключей и восстановить все возможные варианты исходного текста, то они все окажутся равновероятными. Не существует способа выбрать исходный текст, который был действительно послан. Теоретически доказано, что одноразовые системы являются нераскрываемыми системами, поскольку их шифртекст не содержит достаточной информации для восстановления открытого текста.

Казалось бы, что благодаря данному достоинству одноразовые системы следует применять во всех случаях, требующих абсолютной информационной безопасности. Однако возможности применения одноразовой системы ограничены чисто практическими аспектами. Существенным моментом является требование одноразового использования случайной ключевой последовательности. Ключевая последовательность с длиной, не меньшей длины сообщения, должна передаваться получателю сообщения заранее или отдельно по некоторому секретному каналу. Это требование не будет слишком обременительным для передачи действительно важных одноразовых сообщений, например, по горячей линии Вашингтон - Москва. Однако такое требование практически неосуществимо для современных систем обработки информации, где требуется шифровать многие миллионы символов.

В некоторых вариантах одноразового блокнота прибегают к более простому управлению ключевой последовательностью, но это приводит к некоторому снижению надежности шифра. Например, ключ определяется указанием места в книге, известной отправителю и получателю сообщения. Ключевая последовательность начинается с указанного места этой книги и используется таким же образом, как в системе Вижинера. Иногда такой

шифр называют шифром с бегущим ключом. Управление ключевой последовательностью в таком варианте шифра намного проще, так как длинная ключевая последовательность может быть представлена в компактной форме. Но с другой стороны, эти ключи не будут случайными. Поэтому у криптоаналитика появляется возможность использовать информацию о частотах букв исходного естественного языка.

Шифрование методом Вернама

Система шифрования Вернама является в сущности частным случаем системы шифрования Вижинера при значении модуля $m = 2$. Конкретная версия этого шифра, предложенная в 1926 г. Гилбертом Вернамом, сотрудником фирмы AT&T США, использует двоичное представление символов исходного текста.

Каждый символ исходного открытого текста из английского алфавита $\{A, B, C, D, \dots, Z\}$, расширенного шестью вспомогательными символами (пробел, возврат каретки и т.п.), сначала кодировался в 5-битовый блок (b_0, b_1, \dots, b_4) телеграфного кода Бодо.

Случайная последовательность двоичных ключей k_0, k_1, k_2, \dots заранее записывалась на бумажной ленте.

Схема передачи сообщений с использованием шифрования методом Вернама показана ниже. Шифрование исходного текста, предварительно преобразованного в последовательность двоичных символов x , осуществлялось путем сложения по модулю 2 символов x с последовательностью двоичных ключей k .

Символы шифртекста

$$y = x \oplus k$$

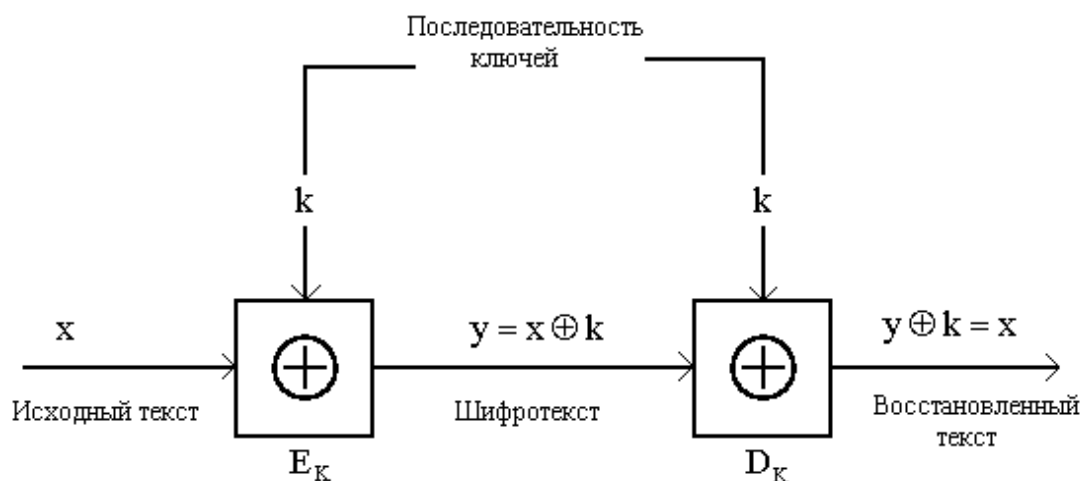


Схема шифрования и расшифрования сообщений по методу Вернама

Расшифрование состоит в сложении по модулю 2 символов y шифртекста с той же последовательностью ключей k :

$$y \oplus k = x \oplus k \oplus k = x.$$

При этом последовательности ключей, использованные при шифровании и расшифровании, компенсируют друг друга (при сложении по модулю 2), и в результате восстанавливаются символы x исходного текста.

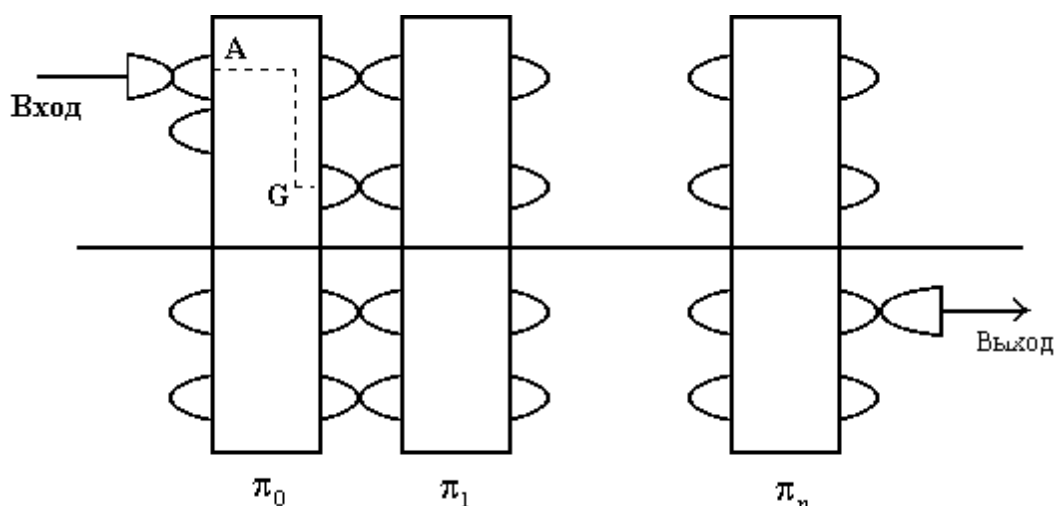
При разработке своей системы Вернам проверял ее с помощью закольцованных лент, установленных на передатчике и приемнике для того, чтобы использовалась одна и та же последовательность ключей.

Следует отметить, что метод Вернама не зависит от длины последовательности ключей и, кроме того, он позволяет использовать случайную последовательность ключей. Однако при реализации метода Вернама возникают серьезные проблемы, связанные с необходимостью доставки получателю такой же последовательности ключей, как у отправителя, либо с необходимостью безопасного хранения идентичных последовательностей ключей у отправителя и получателя. Эти недостатки системы шифрования Вернама преодолены при шифровании методом гаммирования.

Роторные машины

В 20-х годах XX века были изобретены электромеханические устройства шифрования, автоматизирующие процесс шифрования. Принцип работы таких машин основан на многоалфавитной замене символов исходного текста по длинному ключу согласно версии шифра Вижинера. Большинство из них - американская машина SIGABA (M-134), английская TYPEX, немецкая ENIGMA, японская PURPLE были роторными машинами.

Главной деталью роторной машины является ротор (или колесо) с проволочными переключками внутри. Ротор имеет форму диска (размером с хоккейную шайбу). На каждой стороне диска расположены равномерно по окружности m электрических контактов, где m - число знаков алфавита (в случае латинского алфавита $m = 26$). Каждый контакт на передней стороне диска соединен с одним из контактов на задней стороне. В результате электрический сигнал, представляющий знак, будет переставлен в соответствии с тем, как он проходит через ротор от передней стороны к задней. Например, ротор можно закоммутировать проволочными переключками для подстановки G вместо A, U вместо B, L вместо C и т.д.



Банк роторов

При повороте ротора из одного положения в другое подстановка, которую он осуществляет в приходящем сигнале, будет изменяться. В общем случае эту подстановку можно записать в виде

$$T = C^j p C^{-j}$$

где p - подстановка, реализуемая ротором в его начальном положении; C - циклический сдвиг на одну позицию; C^j - циклический сдвиг на j позиций.

Например, если начальная подстановка ротора $p(A) = G$ и ротор сдвигается на три позиции ($j = 3$), то открытый текст D будет против того контакта ротора, который используется

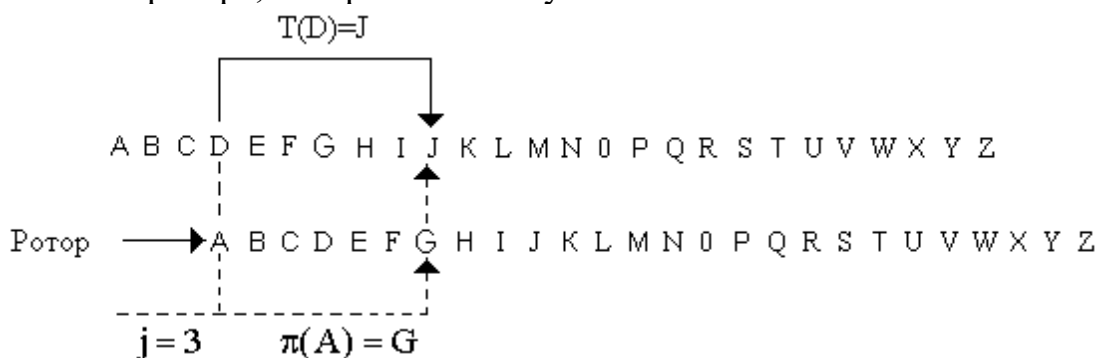


Схема формирования подстановки при сдвиге ротора ($j=3$) для представления открытого текста A , а шифрованный текст J окажется против того контакта ротора, который используется для представления шифрованного текста G , и результирующая подстановка $T(D) = G$ при $j = 3$. Алгебраически это записывается в виде

$$T(D) = C^3 p C^{-3}(D) = C^3 p(A) = C^3(G) = J.$$

Роторы можно объединить в банк роторов таким образом, чтобы выходные контакты одного ротора касались входных контактов следующего ротора. При этом электрический импульс от нажатой клавиши с буквой исходного текста, входящий с одного конца банка роторов, будет переставляться каждым из роторов, до тех пор, пока не покинет банк.

Такой банк может реализовывать большое число подстановок, соответствующих различным комбинациям положений роторов. Для получения сильной криптографической системы расположение роторов должно меняться при переходе от знака к знаку сообщения.

Роторная машина состоит из банка роторов и механизма для изменения положения роторов с каждым зашифрованным знаком, объединенного с устройствами ввода и вывода, такими как устройство считывания с перфоленты и печатающее устройство.

Простейшее из возможных движений ротора - это движение по принципу одометра; оно использовалось в немецкой машине Enigma во время второй мировой войны. При шифровании одного знака правое крайнее колесо поворачивается на одну позицию. Когда это (и любое другое) колесо переместится на m позиций и совершит полный оборот, колесо, расположенное слева от него, передвинется на одну позицию, и процесс будет повторяться. Этот процесс проведет банк роторов сквозь все его возможные положения, прежде чем цикл повторится. Поскольку все роторы перемещаются с разными скоростями, период n -роторной машины составляет 26^n (при $m = 26$).

Для закона движения ротора желательны следующие характеристики:

- период должен быть большим;
- после шифрования каждого знака все роторы или большая их часть должны повернуться друг относительно друга.

Движение по принципу одометра оптимально в смысле первого требования, но совершенно неудовлетворительно в отношении второго требования. Улучшение движения по принципу одометра можно получить, если поворачивать каждый ротор более чем на одну позицию. Если смещения каждого ротора не имеют общих множителей с объемом алфавита m , то период останется максимальным.

Другое решение заключается в ограничении числа допустимых остановочных мест для каждого ротора за счет введения внешнего фиксирующего кольца, на котором определенным способом зафиксированы места остановок. При использовании латинского алфавита можно заставить машины поворачиваться и останавливаться следующим образом. Первому колесу разрешается останавливаться в каждой из 26 позиций, второму колесу - только в 25 позициях, третьему колесу - только в 23 позициях, и так далее до шестого колеса, которому разрешается останавливаться только в 17 позициях. Период такой роторной машины теперь составляет 101 млн, а не $26^6 \gg 309$ млн, как в случае движения по принципу одометра. Потеря в длине периода с успехом окупается полученной сложностью движения роторов. Теперь второе требование удовлетворяется довольно хорошо, поскольку каждое из колес перемещается после шифрования каждого знака и многие колеса могут двигаться друг относительно друга.

Роторная машина может быть настроена по ключу изменением любых ее переменных:

- роторов;
- порядка расположения роторов;
- числа мест остановки на колесо;
- характера движения и т.д.

Поскольку перекоммутировать роторы трудно, то обычно на практике машины обеспечивали комплектом роторов, в котором находилось больше роторов, чем можно одновременно поместить в машину. Первичная настройка по ключу производилась выбором роторов, составляющих комплект. Вторичная настройка по ключу производилась выбором порядка расположения роторов в машине и установкой параметров, управляющих движением машины. С целью затруднения расшифрования шифртекстов противником роторы ежедневно переставляли местами или заменяли. Большая часть ключа определяла начальные положения роторов ($26^3=17576$ возможных установок) и конкретные перестановки на коммутационной доске, с помощью которой осуществлялась начальная перестановка исходного текста до его шифрования ($26!=4 \cdot 10^{26}$ возможностей).

Роторные машины были самыми важными криптографическими устройствами во время второй мировой войны и доминировали по крайней мере до конца 50-х годов.

Варианты заданий:

- 1) Реализовать шифрование текстового сообщения, используя любой нестандартный метод перестановки, например, менять местами четные и нечетные символы. Использование данного метода в этом варианте лабораторной работы не запрещается, но желательно проявить инициативу.
- 2) Реализовать шифрование текстового сообщения, используя метод шифрования с подстановкой степени n .
- 3) Реализовать шифрование текстового сообщения, используя метод шифрования с любой маршрутной перестановкой.
- 4) Реализовать шифрование текстового сообщения, используя шифр «Сци-тала».
- 5) Реализовать шифрование текстового сообщения, используя шифр «Поворотная решетка».
- 6) Реализовать шифрование текстового сообщения, используя шифр вертикальной перестановки.
- 7) Реализовать шифрование текстового сообщения, используя шифр «Полибианский квадрат».
- 8) Реализовать шифрование текстового сообщения, используя систему шифрования Цезаря.
- 9) Реализовать шифрование текстового сообщения, используя аффинную систему подстановок Цезаря.
- 10) Реализовать шифрование текстового сообщения, используя систему Цезаря с ключевым словом.
- 11) Реализовать шифрование текстового сообщения, используя шифрующие таблицы Трисемуса.
- 12) Реализовать шифрование текстового сообщения, используя биграммный шифр Плейфейра.
- 13) Реализовать шифрование текстового сообщения, используя систему омофонов.

- 14) Реализовать шифрование текстового сообщения, используя шифр Гронсфелда.
- 15) Реализовать шифрование текстового сообщения, используя систему шифрования Вижинера .
- 16) Реализовать шифрование текстового сообщения, используя шифр «двойной квадрат» Уитстона.
- 17) Реализовать шифрование текстового сообщения, используя метод одноразовой системы шифрования.
- 18) Реализовать шифрование текстового сообщения, используя шифрование методом Вернама.
- 19) Реализовать шифрование текстового сообщения, используя шифр «Роторные машины».

Лабораторная работа №2. Алгоритм RSA

Цель работы: Получить практические навыки по использованию ассиметричных алгоритмов шифрования, на примере использования алгоритма RSA.

Теоретические сведения

Криптосистема RSA

Алгоритм RSA (R.Rivest, A.Shamir, L.Adleman) был предложен еще в 1977 году. С тех пор он весьма упорно противостоит различным атакам, и сейчас является самым распространенным криптоалгоритмом в мире. Он входит во многие криптографические стандарты, используется во многих приложениях и секретных протоколах (включая PEM, S-HTTP и SSL).

Основные принципы работы RSA

Сначала пара математических определений. Целое число называют простым, если оно делится нацело только на единицу и на само себя, иначе его называют составным. Два целых числа называют взаимно простым, если их наибольший общий делитель (НОД) равен 1.

Алгоритм работы RSA таков. Сначала надо получить открытый и секретный ключи:

1. Выбираются два простых числа p и q
2. Вычисляется их произведение $n(=p*q)$
3. Выбирается произвольное число e ($e < n$), такое, что $\text{НОД}(e, (p-1)(q-1)) = 1$, то есть e должно быть взаимно простым с числом $(p-1)(q-1)$.
4. Методом Евклида решается в целых числах уравнение $e*d + (p-1)(q-1)*y = 1$. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d, y) , каждая из которых является решением уравнения в целых числах.
5. Два числа (e, n) – публикуются как открытый ключ.
6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e, n) .

Как же производится собственно шифрование с помощью этих чисел:

1. Отправитель разбивает свое сообщение на блоки, равные $k = \lfloor \log_2(n) \rfloor$ бит, где квадратные скобки обозначают взятие целой части от дробного числа.
2. Подобный блок, как Вы знаете, может быть интерпретирован как число из диапазона $(0; 2^k - 1)$. Для каждого такого числа (назовем его m_i) вычисляется выражение $c_i = ((m_i)^e) \bmod n$. Блоки c_i и есть зашифрованное сообщение Их можно спокойно передавать по открытому каналу, поскольку операция возведения в степень по модулю простого числа, является необратимой математической задачей. Обратная ей

задача носит название «логарифмирование в конечном поле» и является на несколько порядков более сложной задачей. То есть даже если злоумышленник знает числа e и n , то по c_i прочесть исходные сообщения m_i он не может никак, кроме как полным перебором m_i .

А вот на приемной стороне процесс дешифрования все же возможен, и поможет нам в этом хранимое в секрете число d . Достаточно давно была доказана теорема Эйлера, частный случай которой утверждает, что если число n представимо в виде двух простых чисел p и q , то для любого x имеет место равенство $(x^{(p-1)(q-1)}) \bmod n = 1$. Для дешифрования RSA-сообщений воспользуемся этой формулой. Возведем обе ее части в степень $(-y)$: $(x^{(-y)(p-1)(q-1)}) \bmod n = 1^{(-y)} = 1$. Теперь умножим обе ее части на x : $(x^{(-y)(p-1)(q-1)+1}) \bmod n = 1 * x = x$.

А теперь вспомним как мы создавали открытый и закрытый ключи. Мы подбирали с помощью алгоритма Евклида d такое, что $e*d + (p-1)(q-1)*y = 1$, то есть $e*d = (-y)(p-1)(q-1) + 1$. А следовательно в последнем выражении предыдущего абзаца мы можем заменить показатель степени на число $(e*d)$. Получаем $(x^{e*d}) \bmod n = x$. То есть для того чтобы прочесть сообщение $c_i = ((m_i)^e) \bmod n$ достаточно возвести его в степень d по модулю n : $((c_i)^d) \bmod n = ((m_i)^{e*d}) \bmod n = m_i$.

На самом деле операции возведения в степень больших чисел достаточно трудоемки для современных процессоров, даже если они производятся по оптимизированным по времени алгоритмам. Поэтому обычно весь текст сообщения кодируется обычным блочным шифром (намного более быстрым), но с использованием ключа сеанса, а вот сам ключ сеанса шифруется как раз асимметричным алгоритмом с помощью открытого ключа получателя и помещается в начало файла.

В 1990-х годах с помощью распределенных вычислений через Internet предпринимались удачные попытки факторизовать некоторые произвольные большие числа. Максимальное факторизованное число имело 140 десятичных разрядов (1999 год), на что ушло около 2000 MY (Mips/Year – годовая работа компьютера мощностью в миллион целочисленных операций в секунду). Учитывая это, сейчас специалисты (Лаборатория RSA, www.rsa.com/rsalabs) рекомендуют использовать минимальную длину ключа n , не менее чем 768 бит (~230 десятичных разрядов) для малосекретной информации, 1024 бит для обычной и 2048 для особо секретной. Используемая в старых продуктах длина ключа в 512 бит (~160 разрядов) уже под угрозой взлома. Известно, что RSA имеет низкую криптостойкость при шифровании коротких блоков. В таких случаях злоумышленник может взять от блока шифротекста корень степени e по модулю n , что в данном случае будет намного быстрее факторизации. Поэтому короткие блоки обязательно надо «набивать» дополнительными битами. Еще один интересный вопрос касается простых чисел. Для построения ключей алгоритму RSA необходимо найти два простых числа. Благо, среди чисел

простые попадают довольно часто: на отрезке от 1 до n примерно $n/\ln(n)$ чисел являются простыми. Поэтому можно просто брать псевдослучайные числа нужной длины и проверять их на простоту. Проверку числа на простоту можно делать двумя способами: «в лоб», перебором всех его делителей (от 2 до округленного корня из n), или с помощью более «хитрых» тестов на делимость. При переборе всех делителей мы гарантированно можем утверждать, что прошедшее такую проверку число является простым. Однако, время работы такой процедуры будет очень велико. Среди «хитрых» тестов надо выделить тест Миллера–Рабина, так как он на сегодняшний день является наиболее лучшим по всем параметрам. Так вот, проверка Миллера–Рабина работает намного быстрее перебора делителей, но в отличие от него, число, выдаваемое им как простое, с некоторой очень маленькой вероятностью может оказаться составным. На практике обычно применяют последовательно несколько разных «хитрых» тестов, минимизируя тем самым вероятность ошибки. Программная реализация RSA работает медленнее примерно на два порядка по сравнению с симметричными алгоритмами. RSA часто используют вместе с каким-нибудь симметричным шифром. При таком способе все сообщения шифруются с помощью более быстрого симметричного алгоритма, а для пересылки сессионного секретного ключа этого симметричного алгоритма используется RSA. Получается вычислительно дешево и сердито.

Проясним использование алгоритма RSA на конкретном примере. Выбираем два простых числа $p=7$; $q=17$ (на практике эти числа во много раз длиннее). В этом случае $n = p \cdot q$ будет равно 119. Теперь необходимо выбрать e , выбираем $e=5$. Следующий шаг связан с формированием числа d так, чтобы $(d \cdot e) \bmod [(p-1)(q-1)] = 1$. $d=77$ (использован расширенный алгоритм Эвклида). d – секретный ключ, а e и n характеризуют открытый ключ. Пусть текст, который нам нужно зашифровать представляется $M=19$. $C = M^e \bmod n$. Получаем зашифрованный текст $C=66$. Этот «текст» может быть послан соответствующему адресату. Получатель дешифрует полученное сообщение, используя $M = C^d \bmod n$ и $C=66$. В результате получается $M=19$.

На практике общедоступные ключи могут помещаться в специальную базу данных. При необходимости послать партнеру зашифрованное сообщение можно сделать сначала запрос его открытого ключа. Получив его, можно запустить программу шифрации, а результат ее работы послать адресату. На использовании общедоступных ключей базируется и так называемая электронная подпись, которая позволяет однозначно идентифицировать отправителя. Сходные средства могут применяться для предотвращения внесения каких-либо корректив в сообщение на пути от отправителя к получателю. Быстродействующие аппаратные 512-битовые модули могут обеспечить скорость шифрования на уровне 64 кбит в сек. Готовятся ИС, способные выполнять такие операции со скоростью 1

Мбайт/сек. Разумный выбор параметра ϵ позволяет заметно ускорить реализацию алгоритма.

Варианты заданий:

- 1) Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двенадцати.
- 2) Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей шестнадцати.
- 3) Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двадцати.
- 4) Выполнить шифрование текстового файла, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двенадцати.
- 5) Выполнить шифрование текстового файла, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей шестнадцати.
- 6) Выполнить шифрование текстового файла, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двадцати.
- 7) Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности p и q для каждого нового шифрования.
- 8) Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей шестнадцати, выполнив условие случайности p и q для каждого нового шифрования.
- 9) Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двадцати, выполнив условие случайности p и q для каждого нового шифрования.
- 10) Выполнить шифрование текстового файла методом, RSA, используя в качестве p и q простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности p и q для каждого нового шифрования.
- 11) Выполнить шифрование текстового файла методом, RSA, используя в качестве p и q простые числа с разрядностью не меньшей шестнадцати, выполнив условие случайности p и q для каждого нового шифрования.
- 12) Выполнить шифрование текстового файла, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двадцати, выполнив условие случайности p и q для каждого нового шифрования.

Лабораторная работа №3. Несимметричные алгоритмы шифрования

Цель работы: Получить практические навыки по использованию несимметричных алгоритмов шифрования, на примере использования алгоритма Диффи-Хеллмана и Эль-Гамала, а так же по применению электронных подписей на примере метода Эль-Гамала.

Теоретические сведения

Алгоритм Диффи-Хеллмана:

Алгоритм назван по фамилиям его создателей Диффи (Diffie) и Хеллмана (Hellman).

Метод помогает обмениваться секретным ключом для симметричных криптосистем, но использует метод, очень похожий на асимметричный алгоритм RSA. Это не симметричный алгоритм, так как для шифрования и дешифрования используются различные ключи. Так же это не схема с открытым ключом, потому что ключи легко получаются один из другого, и ключ шифрования и ключ дешифрования должны храниться в секрете.

Определим круг его возможностей. Предположим, что двум абонентам необходимо провести конфиденциальную переписку, а в их распоряжении нет первоначально оговоренного секретного ключа. Однако, между ними существует канал, защищенный от модификации, то есть данные, передаваемые по нему, могут быть прослушаны, но не изменены (такие условия имеют место довольно часто). В этом случае две стороны могут создать одинаковый секретный ключ, ни разу не передав его по сети, по следующему алгоритму.

Предположим, что обоим абонентам известны некоторые два числа v и q . Они, впрочем, известны и всем остальным заинтересованным лицам. Например, они могут быть просто фиксированно «защиты» в программное обеспечение. Далее один из партнеров $P1$ генерирует случайное или псевдослучайное простое число x и посылает другому участнику будущих обменов $P2$ значение $A = q^x \bmod n$

По получении A партнер $P2$ генерирует случайное или псевдослучайное простое число y и посылает $P2$ вычисленное значение $B = q^y \bmod n$

Партнер $P1$, получив B , вычисляет $Kx = B^x \bmod n$, а партнер $P2$ вычисляет $Ky = A^y \bmod n$. Алгоритм гарантирует, что числа Ky и Kx равны и могут быть использованы в качестве секретного ключа для шифрования. Ведь даже перехватив числа A и B , трудно вычислить Kx или Ky .

Например, по вычисленным $Kx = Ky = K$ абоненты могут зашифровать сообщение $M=123$ по следующему алгоритму: к каждому символу сообщения M добавить $K \Rightarrow$ сообщение $C=234$, при $K=1$. Соответственно алгоритмом расшифрования будет разность ключа K из каждого символа сообщения C .

Пример:

Пусть

$n=3; q=5;$

$x=5; y=7;$

тогда $A = q^x \bmod n = 1$, а $B = q^y \bmod n = 2$, то вычислив $Kx = B^x \bmod n$ и $Ky = A^y \bmod n$ получим $Kx = Ky = 1$. Зашифруем приведенное выше сообщение $M=123$ по приведенному выше алгоритму \Rightarrow сообщение $C=234$, расшифровав сообщение C по обратному алгоритму получим сообщение $M=123$.

Необходимо еще раз отметить, что алгоритм Диффи-Хеллмана работает только на линиях связи, надежно защищенных от модификации. Если бы он был применим на любых открытых каналах, то давно снял бы проблему распространения ключей и, возможно, заменил собой всю асимметричную криптографию. Однако, в тех случаях, когда в канале возможна модификация данных, появляется очевидная возможность вклинивания в процесс генерации ключей «злоумышленника-посредника» по той же самой схеме, что и для асимметричной криптографии.

Алгоритм Эль-Гамала:

Алгоритм Эль-Гамала может использоваться для формирования электронной подписи или для шифрования данных. Он базируется на трудности вычисления дискретного логарифма. Для генерации пары ключей сначала берется простое число p и два случайных простых числа g и x , каждое из которых меньше p . Затем вычисляется:

$$y = g^x \bmod p$$

Общедоступными ключами являются y , g и p , а секретным ключом является x . Для подписи сообщения M выбирается случайное число k , которое является простым по отношению к $p-1$. После этого вычисляется $a = g^k \bmod p$. Далее из уравнения $M = (xa + kb) \bmod (p-1)$ находим b . Электронной подписью для сообщения M будет служить пара a и b . Случайное число k следует хранить в секрете. Для верификации подписи необходимо проверить равенство:

$$y^a a^b \bmod p = g^M \bmod p.$$

Пример:

Выберем $p=11$, $g=2$, а закрытый ключ $x=8$.

Вычислим $y = g^x \bmod p = 3$.

Открытым ключом являются $p=11$, $g=2$, $y=3$, чтобы подписать $M=5$ сначала выберем случайное число $k=9$. Вычисляем $a = g^k \bmod p = 2^9 \bmod 11 = 6$ и с помощью расширенного алгоритма Эвклида находим:

$$M = (xa + kb) \bmod (p - 1)$$

$$5 = (8*6 + 9 * b) \bmod 10$$

Решение: $b=3$, а подпись представляет собой пару $a=6$ и $b=3$.

Шифрование Эль-Гамала:

Модификация алгоритма позволяет шифровать сообщения. Для шифрования сообщения M сначала выбирается случайное число k , взаимно простое с $p-1$, затем вычисляются

$$a = g^k \bmod p$$

$$b = y^k M \bmod p$$

Пара a и b представляют собой зашифрованный текст. Следует заметить, что зашифрованный текст имеет размер в два раза больше исходного. Для дешифрования производится вычисление:

$$M = b/a^x \bmod p.$$

Пример:

Вернемся к предыдущему примеру:

$$a=6, b = y^k M \bmod p = 3.$$

$$\text{Расшифруем: } M = b/a^x \bmod p = 5.$$

Варианты заданий:

- 13) Выполнить шифрование строки исходного текста, методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, используя в алгоритме шифрования функцию возведения в степень.
- 14) Выполнить шифрование строки исходного текста, методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, используя в алгоритме шифрования функцию тангенса.
- 15) Выполнить шифрование строки исходного текста, методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати, используя в алгоритме шифрования функцию тангенса.
- 16) Выполнить шифрование строки исходного текста, методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати, используя в алгоритме шифрования функцию возведения в степень.
- 17) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати.
- 18) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати.
- 19) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, используя в алгоритме шифрования функцию возведения в степень.
- 20) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, используя в алгоритме шифрования функцию тангенса.
- 21) Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати.
- 22) Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p не менее двадцати.
- 23) Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати.

- 24) Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати.
- 25) Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p не менее двадцати.
- 26) Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати.
- 27) Выполнить шифрование строки исходного текста методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности x и y для каждого нового шифрования и используя в алгоритме шифрования функцию возведения в степень.
- 28) Выполнить шифрование строки исходного текста, методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности x и y для каждого нового шифрования и используя в алгоритме шифрования функцию тангенса.
- 29) Выполнить шифрование строки исходного текста методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати, выполнив условие случайности x и y для каждого нового шифрования и используя в алгоритме шифрования функцию тангенса.
- 30) Выполнить шифрование строки исходного текста, методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати, выполнив условие случайности x и y для каждого нового шифрования и используя в алгоритме шифрования функцию возведения в степень.
- 31) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности x и y для каждого нового шифрования.
- 32) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати, выполнив условие случайности x и y для каждого нового шифрования.
- 33) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двадцати, выполнив условие случайности x и y для каж-

дого нового шифрования и используя в алгоритме шифрования функцию тангенса.

- 34) Выполнить шифрование текстового файла методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности x и y для каждого нового шифрования и используя в алгоритме шифрования функцию возведения в степень.
- 35) Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности x и g для каждого нового шифрования.
- 36) Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p не менее двадцати, выполнив условие случайности x и g для каждого нового шифрования.
- 37) Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати, выполнив условие случайности x и g для каждого нового шифрования.
- 38) Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности x и g для каждого нового шифрования.
- 39) Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p не менее двадцати, выполнив условие случайности x и g для каждого нового шифрования.
- 40) Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати, выполнив условие случайности x и g для каждого нового шифрования.

Лабораторная работа №4. Блочные шифры

Цель работы: Получить практические навыки по созданию и применению блочных шифров.

Теоретические сведения

Характерной особенностью блочных криптоалгоритмов является тот факт, что в ходе своей работы они производят преобразование блока входной информации фиксированной длины и получают результирующий блок того же объема, но недоступный для прочтения сторонним лицам, не владеющим ключом. Таким образом, схему работы блочного шифра можно описать функциями $Z = \text{Encrypt}(X, \text{Key})$ и $X = \text{Decrypt}(Z, \text{Key})$

Ключ *Key* является параметром блочного криптоалгоритма и представляет собой некоторый блок двоичной информации фиксированного размера. Исходный (*X*) и зашифрованный (*Z*) блоки данных также имеют фиксированную разрядность, равную между собой, но необязательно равную длине ключа.

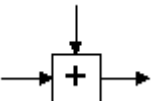
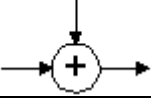
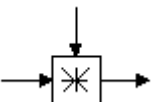
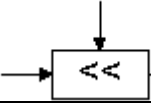
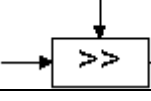
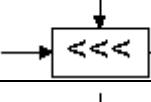

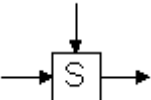
На функцию стойкого блочного шифра $Z = \text{Encrypt}(X, \text{Key})$ накладываются следующие условия:

- Функция *Encrypt* должна быть обратимой.
- Не должно существовать иных методов прочтения сообщения *X* по известному блоку *Z*, кроме как полным перебором ключей *Key*.
- Не должно существовать иных методов определения каким ключом *Key* было произведено преобразование известного сообщения *X* в сообщение *Z*, кроме как полным перебором ключей.

Все действия, производимые над данными блочным криптоалгоритмом, основаны на том факте, что преобразуемый блок может быть представлен в виде целого неотрицательного числа из диапазона, соответствующего его разрядности. Так, например, 32-битный блок данных можно интерпретировать как число из диапазона 0..4'294'967'295. Кроме того, блок, разрядность которого обычно является «степенью двойки», можно трактовать как несколько независимых неотрицательных чисел из меньшего диапазона (рассмотренный выше 32-битный блок можно также представить в виде 2 независимых чисел из диапазона 0..65535 или в виде 4 независимых чисел из диапазона 0..255).

Над этими числами блочным криптоалгоритмом и производятся по определенной схеме следующие действия (слева даны условные обозначения этих операций на графических схемах алгоритмов) :

Биективные математические функции

	Сложение	$X' = X + V$
	Исключающее ИЛИ	$X' = X \text{ XOR } V$
	Умножение по модулю $2^N + 1$	$X' = (X * V) \bmod (2^N + 1)$
	Умножение по модулю 2^N	$X' = (X * V) \bmod (2^N)$
Битовые сдвиги		
	Арифметический сдвиг влево	$X' = X \text{ SHL } V$
	Арифметический сдвиг вправо	$X' = X \text{ SHR } V$
	Циклический сдвиг влево	$X' = X \text{ ROL } V$
	Циклический сдвиг вправо	$X' = X \text{ ROR } V$
Табличные подстановки		
	S-box (англ. substitute)	$X' = \text{Table}[X, V]$

В качестве параметра V для любого из этих преобразований может использоваться:

фиксированное число (например, $X' = X + 125$)

число, получаемое из ключа (например, $X' = X + F(\text{Key})$)

число, получаемое из независимой части блока (например, $X_2' = X_2 + F(X_1)$)

Последний вариант используется в схеме, названной по имени ее создателя сетью Фейштеля (нем. Feistel).

Сеть Фейштеля

Суть метода - смешивания текущей части шифруемого блока с результатом некоторой функции, вычисленной от другой независимой части того же блока. Эта методика получила широкое распространение, поскольку обеспечивает выполнение требования о многократном использовании ключа и материала исходного блока информации.

Классическая сеть Фейштеля имеет следующую структуру:

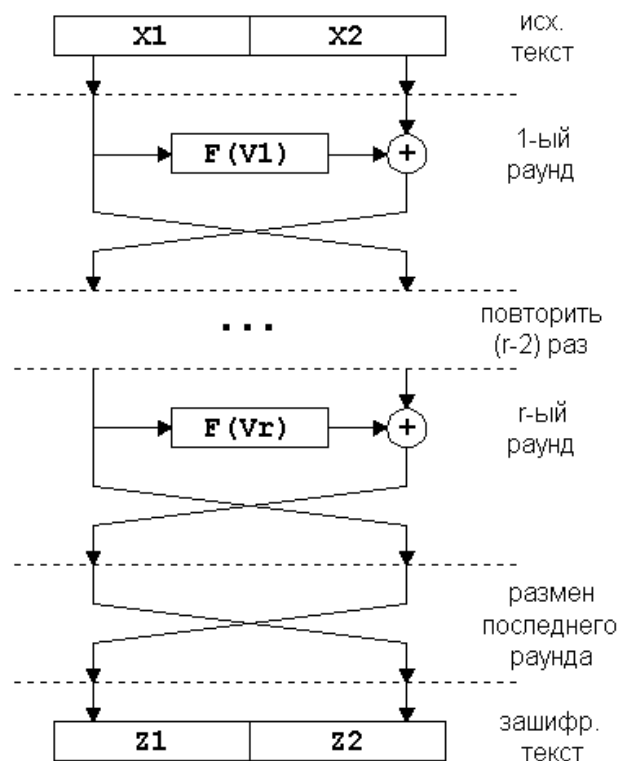


Рис.1.

Независимые потоки информации, порожденные из исходного блока, называются ветвями сети. В классической схеме их две. Величины V_i именуются параметрами сети, обычно это функции от материала ключа. Функция F называется образующей. Действие, состоящее из однократного вычисления образующей функции и последующего наложения ее результата на другую ветвь с обменом их местами, называется циклом или раундом (англ. round) сети Фейштеля. Оптимальное число раундов K – от 8 до 32. Важно то, что увеличение количества раундов значительно увеличивает криптоскойстость любого блочного шифра к криптоанализу. Возможно, эта особенность и повлияла на столь активное распространение сети Фейштеля – ведь при обнаружении, скажем, какого-либо слабого места в алгоритме, почти всегда достаточно увеличить количество раундов на 4-8, не переписывая сам алгоритм. Часто количество раундов не фиксируется разработчиками алгоритма, а лишь указываются разумные пределы (обязательно нижний, и не всегда – верхний) этого параметра.

Сеть Фейштеля обладает тем свойством, что даже если в качестве образующей функции F будет использовано необратимое преобразование, то и в этом случае вся цепочка будет восстанавливаема. Это происходит вследствие того, что для обратного преобразования сети Фейштеля не нужно вычислять функцию F^{-1} .

Более того, как нетрудно заметить, сеть Фейштеля симметрична. Использование операции XOR, обратимой своим же повтором, и инверсия последнего обмена ветвей делают возможным раскодирование блока той же сетью Фейштеля, но с инверсным порядком параметров V_i . Заметим,

что для обратимости сети Фейштеля не имеет значение является ли число раундов четным или нечетным числом. В большинстве реализаций схемы, в которых оба вышеперечисленные условия (операция XOR и уничтожение последнего обмена) сохранены, прямое и обратное преобразования производятся одной и той же процедурой, которой в качестве параметра передается вектор величин V_i либо в исходном, либо в инверсном порядке.

С незначительными доработками сеть Фейштеля можно сделать и абсолютно симметричной, то есть выполняющей функции шифрования и дешифрования одним и тем же набором операций. Математическим языком это записывается как "Функция EnCrypt тождественно равна функции DeCrypt". Если мы рассмотрим граф состояний криптоалгоритма, на котором точками отмечены блоки входной и выходной информации, то при каком-то фиксированном ключе для классической сети Фейштеля мы будем иметь картину, изображенную на рис.2а, а во втором случае каждая пара точек получит уникальную связь, как изображено на рис. 2б. Модификация сети Фейштеля, обладающая подобными свойствами приведена на рисунке 3. Как видим, основная ее хитрость в повторном использовании данных ключа в обратном порядке во второй половине цикла. Необходимо заметить, однако, что именно из-за этой недостаточно исследованной специфики такой схемы (то есть потенциальной возможности ослабления зашифрованного текста обратными преобразованиями) ее используют в криптоалгоритмах с большой осторожностью.

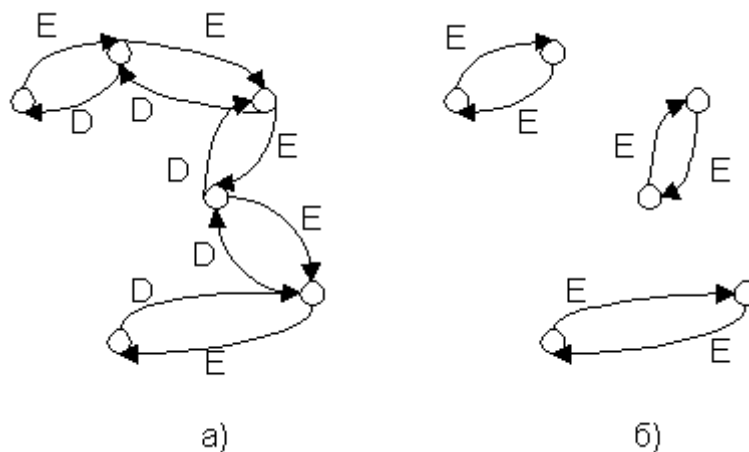


Рис.2.

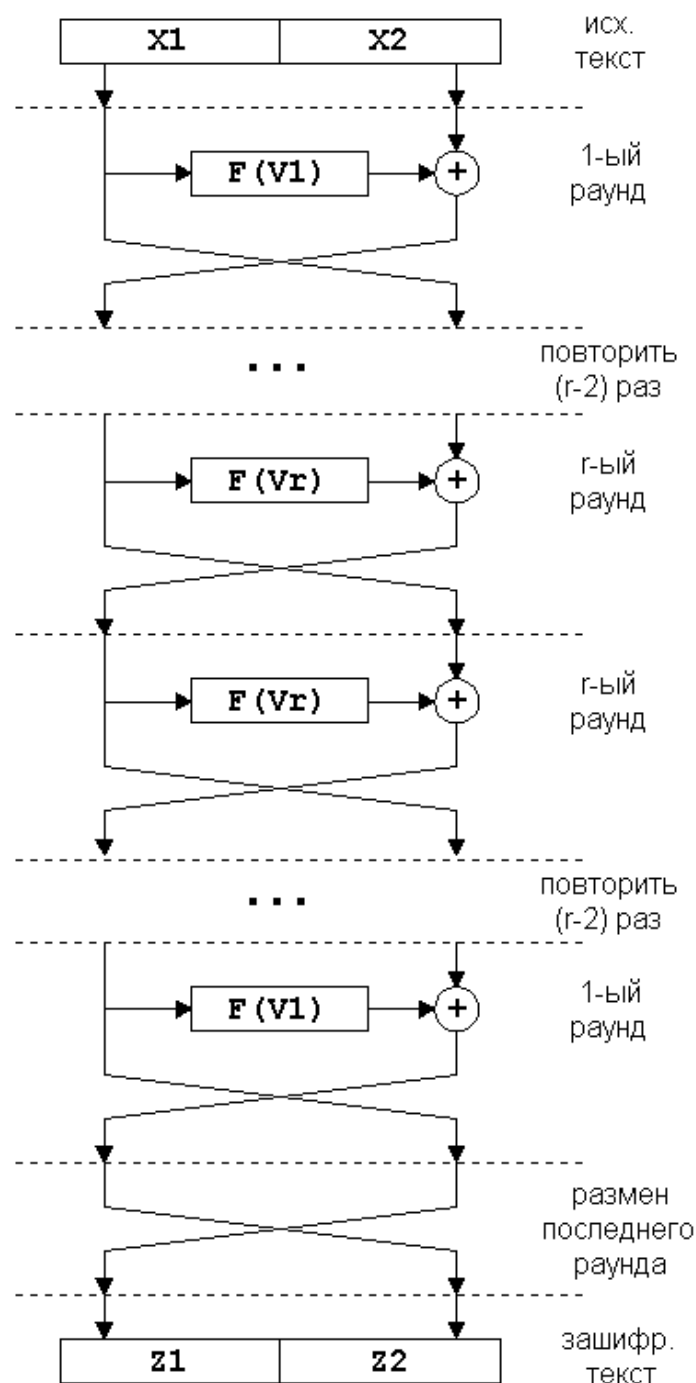


Рис.3.

А вот модификацию сети Фейстеля для большего числа ветвей применяют гораздо чаще. Это в первую очередь связано с тем, что при больших размерах кодируемых блоков (128 и более бит) становится неудобно работать с математическими функциями по модулю 64 и выше. Как известно, основные единицы информации обрабатываемые процессорами на сегодняшний день – это байт и двойное машинное слово 32 бита. Поэтому все чаще и чаще в блочных криптоалгоритмах встречается сеть Фейстеля с 4-мя ветвями. Самый простой принцип ее модификации изображен на рисунке 4а. Для более быстрого перемешивания информации между ветвями (а это основная проблема сети Фейстеля с большим количеством

ветвей) применяются две модифицированные схемы, называемые «type-2» и «type-3». Они изображены на рисунках 4б и 4в.

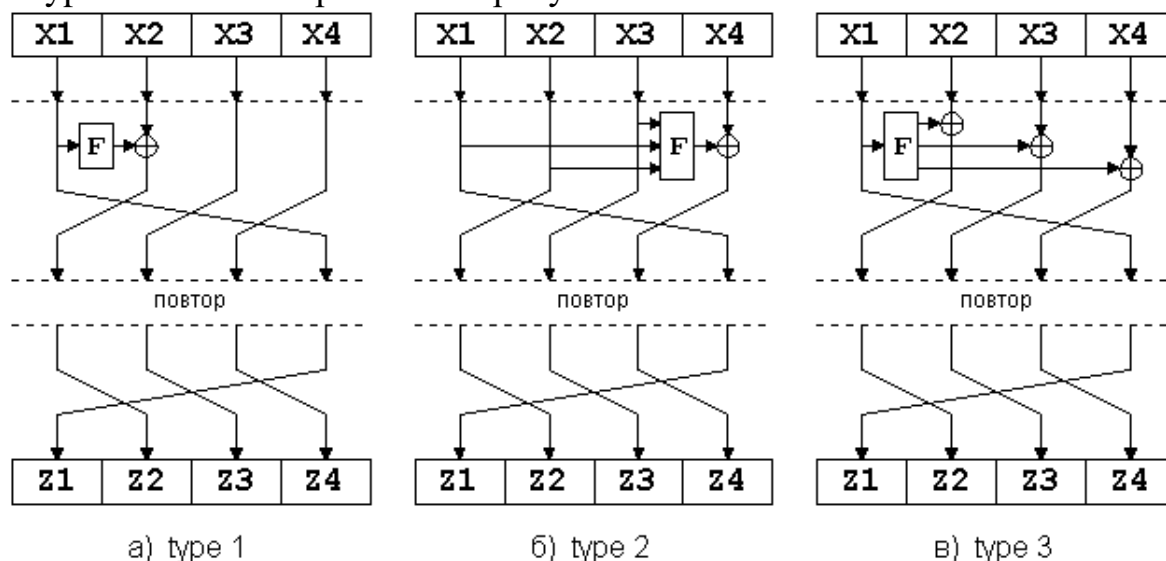


Рис.4.

Сеть Фейштеля надежно зарекомендовала себя как криптостойкая схема произведения преобразований, и ее можно найти практически в любом современном блочном шифре. Незначительные модификации касаются обычно дополнительных начальных и конечных преобразований (англоязычный термин – whitening) над шифруемым блоком. Подобные преобразования, выполняемые обычно также либо «исключающим ИЛИ» или сложением имеют целью повысить начальную рандомизацию входного текста. Таким образом, криптостойкость блочного шифра, использующего сеть Фейштеля, определяется на 95% функцией F и правилом вычисления V_i из ключа. Эти функции и являются объектом все новых и новых исследований специалистов в области криптографии.

Пример блочного шифра - Блочный шифр DES

Стандарт шифрования DES (Data Encryption Standard) был разработан в 1970-х годах, он базируется на алгоритме DEA.

Исходные идеи алгоритма шифрования данных DEA (data encryption algorithm) были предложены компанией IBM еще в 1960-х годах и базировались на идеях, описанных Клодом Шенноном в 1940-х годах. Первоначально эта методика шифрования называлась lucifer (разработчик Хорст Фейштель, название dea она получила лишь в 1976 году. Lucifer был первым блочным алгоритмом шифрования, он использовал блоки размером 128 бит и 128-битовый ключ. По существу этот алгоритм являлся прототипом DEA. В 1986 в Японии (NIT) разработан алгоритм FEAL(Fast data Encipherment ALgorithm), предназначенный для использования в факсах,

модемах и телефонах (длина ключа до 128 бит). Существует и ряд других разработок.

DEA оперирует с блоками данных размером 64 бита и использует ключ длиной 56 бит. Такая длина ключа соответствует 10^{17} комбинаций, что обеспечивало до недавнего времени достаточный уровень безопасности. В дальнейшем можно ожидать расширения ключа до 64 бит (например, LOKI) или вообще замены DES другим стандартом, например Советский шифр ГОСТ – 28147-89, алгоритм работы которого очень схож с DES использует так же 64-битные блоки, но 256-битный ключ.

Входной блок данных, состоящий из 64 бит, преобразуется в выходной блок идентичной длины. Ключ шифрования должен быть известен, как отправляющей так и принимающей сторонам. В алгоритме широко используются перестановки битов текста.

Вводится функция f , которая работает с 32-разрядными словами исходного текста (A) и использует в качестве параметра 48-разрядный ключ (J). Схеме работы функции f показана на рис. 5. Сначала 32 входные разряда расширяются до 48, при этом некоторые разряды повторяются. Схема этого расширения показана ниже (номера соответствуют номерам бит исходного 32-разрядного кода).

32 1 2 3 4 5
4 5 6 7 8 9
8 9 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
28 29 30 31 32 1

Для полученного 48-разрядного кода и ключа выполняется операция исключающее ИЛИ (XOR). Результирующий 48-разрядный код преобразуется в 32-разрядный с помощью S -матриц. На выходе S -матриц осуществляется перестановка согласно схеме показанной ниже (числа представляют собой порядковые номера бит).

16 7 20 21
29 12 28 17
1 15 23 26
5 18 31 10
2 8 24 14
32 27 3 9
19 13 30 6
22 11 4 25

Преобразование начинается с перестановки бит (**IP – Initial Permutation**) в 64-разрядном блоке исходных данных. 58-ой бит становит-

ся первым, 50-ый – вторым и т.д. Схема перестановки битов показана ниже.

58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7

Полученный блок делится на две 32-разрядные части L_0 и R_0 . Далее 16 раз повторяются следующие 4 процедуры:

Преобразование ключа с учетом номера итерации i (перестановки разрядов с удалением 8 бит, в результате получается 48-разрядный ключ)

Пусть $A=L_i$, а J – преобразованный ключ. С помощью функции $f(A,J)$ генерируется 32-разрядный выходной код.

Выполняется операция XOR для $R_i f(A,J)$, результат обозначается R_{i+1} .

Выполняется операция присвоения $L_{i+1}=R_i$.

Инверсная перестановка разрядов предполагает следующее размещение 64 бит зашифрованных данных (первым битом становится 40-ой, вторым 8-ой и т.д.).

40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9 49 17 57 25

S-матрицы представляют собой таблицы содержащие 4-ряда и 16 столбцов. Матрица $S(1)$ представлена ниже (эта матрица, также как и те, что приведены в ссылке 2, являются рекомендуемыми).

№. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
1 0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
2 4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
3 15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13

Исходный 48-разрядный код делится на 8 групп по 6 разрядов. Первый и последний разряд в группе используется в качестве адреса строки, а средние 4 разряда – в качестве адреса столбца. В результате каждые 6 бит кода преобразуются в 4 бита, а весь 48-разрядный код в 32-разрядный (для этого нужно 8 S-матриц). Существуют разработки, позволяющие выпол-

нять шифрование в рамках стандарта DES аппаратным образом, что обеспечивает довольно высокое быстродействие.

Преобразования ключей K_n ($n=1, \dots, 16$; $K_n = KS(n, key)$, где n – номер итерации) осуществляются согласно алгоритму, показанному на рис. 7.

Для описания алгоритма вычисления ключей K_n (функция KS) достаточно определить структуру «Выбора 1» и «Выбора 2», а также задать схему сдвигов влево. «Выбор 1» и «Выбор 2» представляют собой перестановки битов ключа. При необходимости биты 8, 16, ..., 64 могут использоваться для контроля четности.

Для вычисления очередного значения ключа таблица делится на две части $C0$ и $D0$. В $C0$ войдут биты 57, 49, 41, ..., 44 и 36, а в $D0$ – 63, 55, 47, ..., 12 и 4. Так как схема сдвигов задана $C1, D1$; Cn, Dn и так далее могут быть легко получены из $C0$ и $D0$. Так, например, $C3$ и $D3$ получаются из $C2$ и $D2$ циклическим сдвигом влево на 2 разряда

РС-1 (Выбор 1)	РС-2 (Выбор 2)
57 49 41 33 25 17 9	14 17 11 24 1 5
1 58 50 42 34 26 18	3 28 15 6 21 10
10 2 59 51 43 35 27	23 19 12 4 26 8
19 11 3 60 52 44 36	16 7 27 20 13 2
63 55 47 39 31 23 15	41 52 31 37 47 55
7 62 54 46 38 30 22	30 40 51 45 33 48
14 6 61 53 45 37 29	44 49 39 56 34 53
21 13 5 28 20 12 4	46 42 50 36 29 32

Номер итерации	Число сдвигов влево
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Пример блочного шифра - Блочный шифр ТЕА.

ТЕА (Tiny Encryption Algorithm) - один из самых простых в реализации, разработанный в Кэмбридже в 1985 году. Параметры шифра: длина блока - 64 бита, длина ключа - 128 бит. Оптимизирован под 32 битные процессоры. Испытан временем и является довольно криптостойким.

В алгоритме использована сеть Фейштеля с двумя ветвями в 32 бита каждая. Образующая функция F обратима. Сеть Фейштеля несимметрична из-за использования в качестве операции наложения не исключающего «ИЛИ», а арифметического сложения.

Недостатком алгоритма является некоторая медлительность, вызванная необходимостью повторять цикл Фейштеля 32 раза (это необходимо для тщательного «перемешивания данных» из-за отсутствия табличных подстановок).

Отличительной чертой криптоалгоритма ТЕА является его размер. Простота операций, отсутствие табличных подстановок и оптимизация под 32-разрядную архитектуру процессоров позволяет реализовать его на языке ASSEMBLER в предельно малом объеме кода. Недостатком алгоритма является некоторая медлительность, вызванная необходимостью повторять цикл Фейштеля 32 раза (это необходимо для тщательного «перемешивания данных» из-за отсутствия табличных подстановок).

Варианты:

- 1) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 16 бит.
- 2) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 32 бит.
- 3) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 16 бит.
- 4) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 32 бит.
- 5) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 64 бит.
- 6) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 64 бит, ключ длиной 32 бит.
- 7) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 16 бит.
- 8) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 32 бит.
- 9) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 16 бит.
- 10) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 32 бит.
- 11) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 64 бит.
- 12) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 64 бит, ключ длиной 32 бит.
- 13) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 16 бит, реализуя в алгоритме шифрования методику DES.

24) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 64 бит, ключ длиной 32 бит, реализуя в алгоритме шифрования методику DES.

25) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 16 бит, реализуя, максимально приближенные к идентичности, алгоритмы шифрования и дешифрования.

26) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 32 бит, реализуя, максимально приближенные к идентичности, алгоритмы шифрования и дешифрования.

27) Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 16 бит, реализуя, максимально приближенные к идентичности, алгоритмы шифрования и дешифрования.

28) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 16 бит, реализуя, максимально приближенные к идентичности, алгоритмы шифрования и дешифрования.

29) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 32 бит, реализуя, максимально приближенные к идентичности, алгоритмы шифрования и дешифрования.

30) Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 32 бит, ключ длиной 16 бит, реализуя, максимально приближенные к идентичности, алгоритмы шифрования и дешифрования.

Лабораторная работа №5. Шифры обнаружения и коррекции ошибок

Цель работы: Получить практические навыки по применению шифров обнаружения и коррекции ошибок.

Теоретические сведения Обнаружение ошибок.

Каналы передачи данных ненадежны, да и само оборудование обработки информации работает со сбоями. По этой причине важную роль приобретают механизмы детектирования ошибок. Ведь если ошибка обнаружена, можно осуществить повторную передачу данных и решить проблему. Если исходный код по своей длине равен полученному коду, обнаружить ошибку передачи не предоставляется возможным.

Простейшим способом обнаружения ошибок является контроль по четности. Обычно контролируется передача блока данных (M бит). Этому блоку ставится в соответствие кодовое слово длиной N бит, причем $N > M$. Избыточность кода характеризуется величиной $1 - M/N$. Вероятность обнаружения ошибки определяется отношением M/N (чем меньше это отношение, тем выше вероятность обнаружения ошибки, но и выше избыточность).

При передаче информации она кодируется таким образом, чтобы с одной стороны характеризовать ее минимальным числом символов, а с другой – минимизировать вероятность ошибки при декодировании получателем. Для выбора типа кодирования важную роль играет так называемое расстояние Хэмминга.

Пусть A и B две двоичные кодовые последовательности равной длины. Расстояние Хэмминга между двумя этими кодовыми последовательностями равно числу символов, которыми они отличаются. Например, расстояние Хэмминга между кодами 00111 и 10101 равно 2.

Можно показать, что для детектирования ошибок в n битах, схема кодирования требует применения кодовых слов с расстоянием Хэмминга не менее $N+1$. Можно также показать, что для исправления ошибок в N битах необходима схема кодирования с расстоянием Хэмминга между кодами не менее $2N+1$. Таким образом, конструируя код, мы пытаемся обеспечить расстояние Хэмминга между возможными кодовыми последовательностями больше, чем оно может возникнуть из-за ошибок.

Широко распространены коды с одиночным битом четности. В этих кодах к каждому M бит добавляется 1 бит, значение которого определяется четностью (или нечетностью) суммы этих M бит. Так, например, для двухбитовых кодов 00, 01, 10, 11 кодами с контролем четности будут 000, 011, 101 и 110. Если в процессе передачи один бит будет передан неверно, четность кода из $M+1$ бита изменится.

Предположим, что частота ошибок (BER) равна $p=10^{-4}$. В этом случае вероятность передачи 8 бит с ошибкой составит $1-(1-p)^8=7,9 \times 10^{-4}$. До-

бавление бита четности позволяет детектировать любую ошибку в одном из переданных битов. Здесь вероятность ошибки в одном из 9 бит равна $9p(1-p)^8$. Вероятность же реализации необнаруженной ошибки составит $1 - (1-p)^9 - 9p(1-p)^8 = 3,6 \times 10^{-7}$. Таким образом, добавление бита четности уменьшает вероятность необнаруженной ошибки почти в 1000 раз. Использование одного бита четности типично для асинхронного метода передачи. В синхронных каналах чаще используется вычисление и передача битов четности как для строк, так и для столбцов передаваемого массива данных. Такая схема позволяет не только регистрировать но и исправлять ошибки в одном из битов переданного блока.

Контроль по четности достаточно эффективен для выявления одиночных и множественных ошибок в условиях, когда они являются независимыми. При возникновении ошибок в кластерах бит метод контроля четности неэффективен и тогда предпочтительнее метод вычисления циклических сумм (CRC). В этом методе передаваемый кадр делится на специально подобранный образующий полином. Дополнение остатка от деления и является контрольной суммой.

CRC

В Ethernet Вычисление crc производится аппаратно. На рис. 1. показан пример реализации аппаратного расчета CRC для образующего полинома $B(x) = 1 + x^2 + x^3 + x^5 + x^7$. В этой схеме входной код приходит слева.

Эффективность CRC для обнаружения ошибок на многие порядки выше простого контроля четности. В настоящее время стандартизовано несколько типов образующих полиномов. Для оценочных целей можно считать, что вероятность невыявления ошибки в случае использования CRC, если ошибка на самом деле имеет место, равна $(1/2)^r$, где r - степень образующего полинома.

CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$

Основная идея алгоритма CRC состоит в представлении сообщения в виде огромного двоичного числа, делении его на другое фиксированное двоичное число и использовании остатка этого деления в качестве контрольной суммы. Получив сообщение, приемник может выполнить аналогичное действие и сравнить полученный остаток с «контрольной суммой» (переданным остатком). Приведем пример:

Предположим, что сообщение состоит из 2 байт. Их можно рассматривать, как двоичное число 0000 0110 0001 0111. Предположим, что ширина регистра контрольной суммы составляет 1 байт, в качестве делителя используется 1001, тогда сама контрольная сумма будет равна остатку от де-

ления 0000 0110 0001 0111 на 1001. Хотя в данной ситуации деление может быть выполнено с использованием стандартных 32 битных регистров, в общем случае это неверно. Поэтому воспользуемся делением "в столбик" в двоичной системе счисления:

1001=9d делитель
 0000011000010111 =0617h =1559d делимое
 0000011000010111/1001=2d остаток

Хотя влияние каждого бита исходного сообщения на частное не столь существенно, однако 4 битный остаток во время вычислений может радикально измениться, и чем больше байтов имеется в исходном сообщении (в делимом), тем сильнее меняется каждый раз величина остатка. Вот почему деление оказывается применимым там, где обычное сложение работать отказывается.

В нашем случае передача сообщения вместе с 4 битной контрольной суммой выглядела бы (в шестнадцатеричном виде) следующим образом: 06172, где 0617 – это само сообщение, а 2 – контрольная сумма. Приемник, получив сообщение, мог бы выполнить аналогичное деление и проверить, равен ли остаток переданному значению (2).

Полиномиальная арифметика

Все CRC алгоритмы основаны на полиномиальных вычислениях, и для любого алгоритма CRC можно указать, какой полином он использует. Что это значит? Вместо представления делителя, делимого (сообщения), частного и остатка в виде положительных целых чисел, можно представить их в виде полиномов с двоичными коэффициентами или в виде строки бит, каждый из которых является коэффициентом полинома. Например, десятичное число 23 в шестнадцатеричной системе счисления имеет вид 17, в двоичном – 10111, что совпадает с полиномом: $1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0$ или, упрощенно: $x^4 + x^2 + x^1 + x^0$

И сообщение, и делитель могут быть представлены в виде полиномов, с которыми, как и раньше можно выполнять любые арифметические действия; только теперь надо не забывать о «иксах». Предположим, что мы хотим перемножить, например, 1101 и 1011. Это можно выполнить, как умножение полиномов:

$$\begin{aligned} &(x^3 + x^2 + x^0)(x^3 + x^1 + x^0) \\ &= (x^6 + x^4 + x^3 + x^5 + x^3 + x^2 + x^3 + x^1 + x^0) = \\ &= x^6 + x^5 + x^4 + 3 \cdot x^3 + x^2 + x^1 + x^0 \end{aligned}$$

Теперь для получения правильного ответа нам необходимо указать, что x равен 2, и выполнить перенос бита от члена $3 \cdot x^3$. В результате получим: $x^7 + x^3 + x^2 + x^1 + x^0$

А то, что если мы считаем, « X » нам не известным, то мы не можем выполнить перенос. Нам не известно, что $3 \cdot x^3$ – это то же самое, что и $x^4 + x^3$, так как мы не знаем, что $x = 2$. В полиномиальной арифметике связи между коэффициентами не установлены и, поэтому, коэффициенты при

каждом члене полинома становятся строго типизированными — коэффициент при x^2 имеет иной тип, чем при x^3 . Если коэффициенты каждого члена полинома совершенно изолированы друг от друга, то можно работать с любыми видами полиномиальной арифметики, просто меняя правила, по которым коэффициенты работают. Одна из таких схем для нас чрезвычайно интересна, а именно, когда коэффициенты складываются по модулю 2 без переноса — то есть коэффициенты могут иметь значения лишь 0 или 1, перенос не учитывается. Это называется «полиномиальная арифметика по модулю 2». Возвращаясь к предыдущему примеру, получим результат: $=x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + x^0$.

Двоичная арифметика без учета переносов

Сфокусируем наше внимание на арифметике, применяемой во время вычисления CRC. Фактически как операция сложения, так и операция вычитания в CRC арифметике идентичны операции «Исключающее ИЛИ» (eXclusive OR –XOR), что позволяет заменить 2 операции первого уровня (сложение и вычитание) одним действием, которое, одновременно, оказывается инверсным самому себе. Определив действие сложения, перейдем к умножению и делению. Умножение, как и в обычной арифметике, считается суммой значений первого сомножителя, сдвинутых в соответствии со значением второго сомножителя. При суммировании используется CRC-сложение. Деление несколько сложнее, хотя вполне интуитивно понятно как его выполнять. Действия аналогичны простому делению в столбик с применением CRC-сложения. Если число A получено умножением числа B, то в CRC арифметике это означает, что существует возможность сконструировать число A из нуля, применяя операцию XOR к числу B, сдвинутому на различное количество позиций. Например, если A равно 0111010110, а B – 11, то мы можем сконструировать A из B следующим способом:

0111010110

=.....11.

+....11....

+...11.....

+..11.....

Однако, если бы A было бы равно 0111010111, то нам бы не удалось составить его с помощью различных сдвигов числа 11. Поэтому что, как говорят в CRC арифметике, оно не делится на B.

Полностью рабочий пример

Чтобы выполнить вычисление CRC, нам необходимо выбрать делитель. Говоря математическим языком, делитель называется генераторным полиномом(generator polynomial), или просто полиномом, и это ключевое слово любого CRC алгоритма. Степень полинома W (Width – ширина, позиция самого старшего единичного бита) чрезвычайно важна, так как от нее зависят все остальные расчеты. Обычно выбирается степень 16 ил

32, т.к. как это облегчает реализацию алгоритма на современных компьютерах. Степень полинома – это действительная позиция старшего бита, например, степень полинома 10011 равна 4 а, не 5. Выбрав полином приступим к расчетам. Это будет простое деление (в терминах CRC арифметики) сообщения на наш полином.

Единственное, что надо будет сделать до начала работы, так это дополнить сообщение W нулевыми битами. Итак, начнем:

Исходное сообщение: 1101011011

Полином: 10011

Сообщение, дополненное W битами: 11010110110000

Теперь просто поделим сообщение на полином, используя правила CRC-арифметики.

$11010110110000 / 10011 = 1100001010$ (частное, оно никого не интересует)

+

$+1110 = \text{остаток} = \text{контрольная сумма!}$

Как правило, контрольная сумма добавляется к сообщению и вместе с ним передается по каналам связи. В нашем случае будет передано следующее сообщение: 11010110111110. На другом конце канала приемник может сделать одно из равноценных действий:

1) Выделить текст собственно сообщения, вычислить для него контрольную сумму (не забыв при этом дополнить сообщение W битами), и сравнить ее с переданной.

2) Вычислить контрольную сумму для всего переданного сообщения (без добавления нулей), и посмотреть, получится ли в результате нулевой остаток. Оба эти варианта совершенно равноправны. Однако отныне мы будем работать со вторым вариантом, которое является математически более правильным. Таким образом, при вычислении CRC необходимо выполнить следующие действия:

- Выбрать степень полинома W и полином G (степени W).
- Добавить к сообщению W нулевых битов. Назовем полученную строку M' .
- Поделим M' на G с использованием правил CRC арифметики. Полученный остаток и будет контрольной суммой.

Выбор полинома

Во-первых, надо отметить, что переданное сообщение T является произведением полинома. Чтобы понять это, обратите внимание, что 1) последние W бит сообщения – это остаток от деления дополненного нулями исходного сообщения на выбранный полином, и 2) сложение равносильно вычитанию, поэтому прибавление остатка дополняет значение сообщения до следующего полного произведения. Теперь смотрите, если сообщение при передаче было повреждено, то мы получим сообщение $T + E$, где E – это вектор ошибки, '+' – это CRC сложение (или операция XOR).

Получив сообщение, приемник делит $T + E$ на G . Так как $T \bmod G = 0, (T+E) \bmod G = E \bmod G$.

Следовательно, качество полинома, который мы выбираем для перехвата некоторых определенных видов ошибок, будет определяться набором произведений G , так как в случае, когда E также является произведением G , такая ошибка выявлена не будет. Следовательно, наша задача состоит в том, чтобы найти такие классы G , произведения которых будут как можно меньше похожи на шумы в канале передачи (которые и вызывают повреждение сообщения). Давайте рассмотрим, какие типы шумов в канале передачи мы можем ожидать. Однобитовые ошибки. Ошибки такого рода означают, что $E=1000...0000$. Мы можем гарантировать, что ошибки этого класса всегда будут распознаны при условии, что в G по крайней мере 2 бита установлены в "1". Любое произведение G может быть сконструировано операциями сдвига и сложения, и, в тоже время, невозможно получить значение с 1 единичным битом сдвигая и складывая величину, имеющую более 1 единичного бит, так как в результате всегда будет присутствовать по крайней мере 2 бита. Двухбитовые ошибки. Для обнаружения любых ошибок вида $100...000100...000$ (то есть когда E содержит по крайней мере 2 единичных бита) необходимо выбрать такое G , которые бы не имело множителей $11, 101, 1001, 10001$, и так далее. В качестве примера приведем полином с единичными битами в позициях 15, 14 и 1, который не может быть делителем ни одно числа меньше $1...1$, где "...32767" нулей. Ошибки с нечетным количеством бит. Мы можем перехватить любые повреждения, когда E имеет нечетное число бит, выбрав полином G таким, чтобы он имел четное количество бит. Чтобы понять это, обратите внимание на то, что

1) CRC умножение является простой операцией XOR постоянного регистрового значения с различными смещениями;

2) XOR – это всего-навсего операция переключения битов; и

3) если Вы применяете в регистре операцию XOR к величине с четным числом битов, четность количеств единичных битов в регистре останется неизменной.

Например, начнем с $E=111$ и попытаемся сбросить все 3 бита в "0" последовательным выполнением операции XOR с величиной 11 и одним из 2 вариантов сдвигов (то есть, " $E=E \text{ XOR } 011$ " и " $E=E \text{ XOR } 110$ "). Это аналогично задаче о перевертывании стаканов, когда за одно действие можно перевернуть одновременно любые два стакана. Большинство популярных CRC полиномов содержат четное количество единичных битов. Пакетные ошибки. Пакетная ошибка выглядит как $E=000...000111...11110000...00$, то есть E состоит из нулей за исключением группы единиц где-то в середине. Эту величину можно преобразовать в $E=(00000...00)(1111111...111)$, где имеется z нулей в левой части и n единиц в правой. Для выявления этих ошибок нам необходимо установить млад-

ший бит G в 1. При этом необходимо, чтобы левая часть не была множителем G. При этом всегда, пока G шире правой части, ошибка всегда будет распознана. Несколько популярных полиномов:

16 битные: (16,12,5,0) [стандарт "X25"]

(16,15,2,0) ["CRC 16"]

32 битные: (32,26,23,22,16,12,11,10,8,7,5,4,2,1,0) [Ethernet]

Коррекция ошибок.

Исправлять ошибки труднее, чем их детектировать или предотвращать. Процедура коррекции ошибок предполагает два совмещенных процесса: обнаружение ошибки и определение места (идентификация сообщения и позиции в сообщении). После решения этих двух задач, исправление тривиально – надо инвертировать значение ошибочного бита. В наземных каналах связи, где вероятность ошибки невелика, обычно используется метод детектирования ошибок и повторной пересылки фрагмента, содержащего дефект. Для спутниковых каналов с типичными для них большими задержками системы коррекции ошибок становятся привлекательными. Здесь используют коды Хэмминга или коды свертки.

Код Хэмминга представляет собой блочный код, который позволяет выявить и исправить ошибочно переданный бит в пределах переданного блока. Обычно код Хэмминга характеризуется двумя целыми числами, например, (11,7) используемый при передаче 7-битных ASCII-кодов. Такая запись говорит, что при передаче 7-битного кода используется 4 контрольных бита ($7+4=11$). При этом предполагается, что имела место ошибка в одном бите и что ошибка в двух или более битах существенно менее вероятна. С учетом этого исправление ошибки осуществляется с определенной вероятностью. Например, пусть возможны следующие правильные коды (все они, кроме первого и последнего, отстоят друг от друга на расстояние 4):

00000000

11110000

00001111

11111111

При получении кода 00000111 не трудно предположить, что правильное значение полученного кода равно 00001111. Другие коды отстоят от полученного на большее расстояние Хэмминга.

Рассмотрим пример передачи кода буквы $s = 0x073 = 1110011$ с использованием кода Хэмминга (11,7).

Позиция бита:	11	10	9	8	7	6	5	4	3	2	1
Значение бита:	1	1	1	*	0	0	1	*	1	*	*

Символами * помечены четыре позиции, где должны размещаться контрольные биты. Эти позиции определяются целой степенью 2 (1, 2, 4, 8 и т.д.). Контрольная сумма формируется путем выполнения операции XOR (исключающее ИЛИ) над кодами позиций ненулевых битов. В данном случае это 11, 10, 9, 5 и 3. Вычислим контрольную сумму:

11 =	1011
10 =	1010
09 =	1001
05 =	0101
03 =	0011
□ □ □	1110

Таким образом, приемник получит код:

Позиция бита:	11	10	9	8	7	6	5	4	3	2	1
Значение бита:	1	1	1	1	0	0	1	1	1	1	0

Просуммируем снова коды позиций ненулевых битов и получим нуль.

11 =	1011
10 =	1010
09 =	1001
08 =	1000
05 =	0101
04 =	0100
03 =	0011
02 =	0010
□ □ □	0000

Ну а теперь рассмотрим два случая ошибок в одном из битов послышки, например, в бите 7 (1 вместо 0) и в бите 5 (0 вместо 1). Просуммируем коды позиций ненулевых бит еще раз.

11 =	1011	11 =	1011
10 =	1010	10 =	1010
09 =	1001	09 =	1001
08 =	1000	08 =	1000
07 =	0111	04 =	0100
05 =	0101	03 =	0011
04 =	0100	02 =	0010
03 =	0011	□ =	0001
02 =	0010		

□□□	0111
-----	------

В обоих случаях контрольная сумма равна позиции бита, переданного с ошибкой. Теперь для исправления ошибки достаточно инвертировать бит, номер которого указан в контрольной сумме. Понятно, что если ошибка произойдет при передаче более чем одного бита, код Хэмминга при данной избыточности окажется бесполезен.

В общем случае код имеет $N=M+C$ бит и предполагается, что не более чем один бит в коде может иметь ошибку. Тогда возможно $N+1$ состояние кода (правильное состояние и n ошибочных). Пусть $M=4$, а $N=7$, тогда слово-сообщение будет иметь вид: $M_4, M_3, M_2, C_3, M_1, C_2, C_1$. Теперь попытаемся вычислить значения C_1, C_2, C_3 . Для этого используются уравнения, где все операции представляют собой сложение по модулю 2:

$$C_1 = M_1 + M_2 + M_4$$

$$C_2 = M_1 + M_3 + M_4$$

$$C_3 = M_2 + M_3 + M_4$$

Для определения того, доставлено ли сообщение без ошибок, вычисляем следующие выражения (сложение по модулю 2):

$$C_{11} = C_1 + M_4 + M_2 + M_1$$

$$C_{12} = C_2 + M_4 + M_3 + M_1$$

$$C_{13} = C_3 + M_4 + M_3 + M_2$$

Результат вычисления интерпретируется следующим образом.

C11	C12	C13	Значение
1	2	4	Позиция бит
0	0	0	Ошибок нет
0	0	1	Бит C3 не верен
0	1	0	Бит C2 не верен
0	1	1	Бит M3 не верен
1	0	0	Бит C1 не верен
1	0	1	Бит M2 не верен
1	1	0	Бит M1 не верен
1	1	1	Бит M4 не верен

Описанная схема легко переносится на любое число n и M .

Число возможных кодовых комбинаций M помехоустойчивого кода делится на n классов, где N – число разрешенных кодов. Разделение на классы осуществляется так, чтобы в каждый класс вошел один разрешенный код и ближайшие к нему (по расстоянию Хэмминга) запрещенные коды. В процессе приема данных определяется, к какому классу принадлежит пришедший код. Если код принят с ошибкой, он заменяется ближайшим разрешенным кодом. При этом предполагается, что кратность ошибки не более q_m .

Можно доказать, что для исправления ошибок с кратностью не более $q_{m\leq/sub}$ кодовое расстояние должно превышать $2q_m$ (как правило, оно выбирается равным $D = 2q_m + 1$). В теории кодирования существуют следующие оценки максимального числа N n -разрядных кодов с расстоянием D .

$d=1$	$n=2^n$
$d=2$	$n=2^{n-1}$
$d=3$	$n \cdot 2^n / (1+n)$
$d=2q+1$	$N \leq 2^n \left(1 + \sum_{i=1}^d C_n^i\right)^{-1}$ <p>(для кода Хэмминга это неравенство превращается в равенство)</p>

В случае кода Хэмминга первые k разрядов используются в качестве информационных, причем

$$k = n - \log(n+1),$$

откуда следует (логарифм по основанию 2), что k может принимать значения 0, 1, 4, 11, 26, 57 и т.д., это и определяет соответствующие коды Хэмминга (3,1); (7,4); (15,11); (31,26); (63,57) и т.д.

Циклические коды

Обобщением кодов Хэмминга являются циклические коды ВСН (Bose-Chadhuri-Носquenghem). Это коды с широким выбором длины и возможностей исправления ошибок. Циклические коды характеризуются полиномом $g(x)$ степени $n-k$, $g(x) = 1 + g_1x + g_2x^2 + \dots + x^{n-k}$. $g(x)$ называется порождающим многочленом циклического кода. Если многочлен $g(x)$ $n-k$ и является делителем многочлена $x^n + 1$, то код $C(g(x))$ является линейным циклическим (n,k) -кодом. Число циклических n -разрядных кодов равно числу делителей многочлена $x^n + 1$.

При кодировании слова все кодовые слова кратны $g(x)$. $g(x)$ определяется на основе сомножителей полинома $x^n + 1$ как:

$$x^n + 1 = g(x)h(x)$$

Например, если $n=7$ (x^7+1), его сомножители $(1 + x + x^3)(1 + x + x^2 + x^4)$, а $g(x) = 1+x + x^3$.

Чтобы представить сообщение $h(x)$ в виде циклического кода, в котором можно указать постоянные места проверочных и информационных символов, нужно разделить многочлен $x^{n-k}h(x)$ на $g(x)$ и прибавить остаток от деления к многочлену $x^{n-k}h(x)$. Привлекательность циклических кодов заключается в простоте аппаратной реализации с использованием сдвиговых регистров.

Пусть общее число бит в блоке равно N , из них полезную информацию несут в себе K бит, тогда в случае ошибки, имеется возможность ис-

править m бит. Таблица 1 содержит зависимость m от N и K для кодов BCH.

Общее число бит N	Число полезных бит M	Число исправляемых бит m
31	26	1
	21	2
	16	3
63	57	1
	51	2
	45	3
127	120	1
	113	2
	106	3

Увеличивая разность $N-M$, можно не только нарастить число исправляемых бит m , но открыть возможность обнаружить множественные ошибки. В таблице 2 приведен процент обнаруживаемых множественных ошибок в зависимости от M и $N-M$.

Число полезных бит M	Число избыточных бит ($n-m$)		
	6	7	8
32	48%	74%	89%
40	36%	68%	84%
48	23%	62%	81%

Другой блочный метод предполагает «продольное и поперечное» контрольное суммирование передаваемого блока. Блок при этом представляется в виде N строк и M столбцов. Вычисляется биты четности для всех строк и всех столбцов, в результате получается два кода, соответственно длиной N и M бит. На принимающей стороне биты четности для строк и столбцов вычисляются повторно и сравниваются с присланными. При выявлении отличия в бите i кода битов четности строк и бите j – кода столбцов, позиция неверного бита оказывается определенной (i,j). Понятно, что если выявится два и более неверных битов в контрольных кодах строк и столбцов, задача коррекции становится неразрешимой. Уязвим этот метод и для двойных ошибок, когда сбой был, а контрольные коды остались корректными.

Применение кодов свертки позволяют уменьшить вероятность ошибок при обмене, даже если число ошибок при передаче блока данных больше 1.

Представление кодовой комбинации в виде многочлена

Описание циклических кодов и их построение удобно проводить с помощью многочленов (или полиномов). Запись комбинации в виде полинома понадобилась для того, чтобы отобразить формализованным способом операцию циклического сдвига исходной кодовой комбинации. Так, n -элементную кодовую комбинацию можно описать полиномом $(n - 1)$ степени, в виде:

$$A_{n-1}(x) = a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_1 x + a_0,$$

где $a_i = \{0, 1\}$, причем $a_i = 0$ соответствуют нулевым элементам комбинации, $a_i = 1$ — ненулевым.

Запишем полиномы для конкретных 4-элементных комбинаций:

$$1101 \Rightarrow A1(x) = x^3 + x^2 + 1$$

$$1010 \Rightarrow A2(x) = x^3 + x$$

Операции сложения и вычитания выполняются по модулю 2. Они являются эквивалентными и ассоциативными:

$$G1(x) + G2(x) \Rightarrow G3(x)$$

$$G1(x) - G2(x) \Rightarrow G3(x)$$

$$G2(x) + G1(x) \Rightarrow G3(x)$$

Примеры

$$G1(x) = x^5 + x^3 + x$$

$$G2(x) = x^4 + x^3 + 1$$

$$G3(x) = G1(x) \Rightarrow G2(x) = x^5 + x^4 + x + 1$$

Операция деления является обычным делением многочленов, только вместо вычитания используется сложение по модулю 2:

$$G1(x) = x^6 + x^4 + x^3$$

$$G2(x) = x^3 + x^2 + 1$$

$$x^6 + x^4 + x^3 \quad | \quad x^3 + x^2 + 1$$

+-----

$$x^6 + x^5 + x^3 \quad | \quad x^3 + x^2$$

$$x^5 + x^4$$

$$x^5 + x^4 + x^2$$

$$x^2$$

Циклический сдвиг кодовой комбинации

Циклический сдвиг кодовой комбинации — перемещение ее элементов справа налево без нарушения порядка их следования, так что крайний левый элемент занимает место крайнего правого.

Основные свойства и само название циклических кодов связаны с тем, что все разрешенные комбинации бит в передаваемом сообщении (кодовые слова) могут быть получены путем операции циклического сдвига некоторого исходного кодового слова.

Допустим, задана исходная кодовая комбинация и соответствующий ей полином:

$$a(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \dots + a_2 x + a_1$$

Умножим $a(x)$ на x :

$$a(x) \cdot x = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x$$

Так как максимальная степень x в кодовой комбинации длиной n не превышает $(n - 1)$, то из правой части полученного выражения для получения исходного полинома необходимо вычесть $a_n(x^n - 1)$. Вычитание $a_n(x^n - 1)$ называется взятием остатка по модулю $(x^n - 1)$.

Сдвиг исходной комбинации на i тактов можно представить следующим образом: $a(x) \cdot x^i - a_n(x^n - 1)$, то есть умножением $a(x)$ на x^i и взятием остатка по модулю $(x^n - 1)$. Взятие остатка необходимо при получении многочлена степени, большей или равной n .

Идея построения циклических кодов базируется на использовании неприводимых многочленов. Неприводимым называется многочлен, который не может быть представлен в виде произведения многочленов низших степеней, т. е. такой многочлен делится только на самого себя или на единицу и не делится ни на какой другой многочлен. На такой многочлен делиться без остатка двучлен $x^n + 1$. Неприводимые многочлены в теории циклических кодов играют роль образующих полиномов.

Возвращаясь к определению циклического кода и учитывая запись операций циклического сдвига кодовых комбинаций, можно записать порождающую матрицу циклического кода в следующем виде:

$$V = \begin{pmatrix} p(x) \\ p(x) \cdot x - C_2(x^n - 1) \\ p(x) \cdot x^2 - C_3(x^n - 1) \\ \dots \\ p(x) \cdot x^{m-1} - C_m(x^n - 1) \end{pmatrix},$$

где $p(x)$ — исходная кодовая комбинация, на базе которой получены все остальные $(m - 1)$ базовые комбинации, $C_i = 0$ или $C_i = 1$ («0», если результирующая степень полинома $p(x) \cdot x^i$ не превосходит $(n - 1)$, «1», если превосходит).

Комбинация $p(x)$ называется порождающей (порождающей, генераторной) комбинацией. Для построения циклического кода достаточно верно выбрать $p(x)$. Затем все остальные кодовые комбинации получаются такими же, как и в групповом коде.

Порождающий полином должен удовлетворять следующим требованиям:

$p(x)$ должен быть ненулевым;

вес $p(x)$ не должен быть меньше минимального кодового расстояния: $v(p(x)) \geq d_{\min}$;

$p(x)$ должен иметь максимальную степень k (k — число избыточных элементов в коде);

$p(x)$ должен быть делителем полинома $(x^n - 1)$.

Выполнение условия 4 приводит к тому, что все рабочие кодовые комбинации циклического кода приобретают свойство делимости на $p(x)$ без остатка. Учитывая это, можно дать другое определение циклического кода. Циклический код — код, все рабочие комбинации которого делятся на порождающий без остатка.

Для определения степени порождающего полинома можно воспользоваться выражением $r \geq \log_2(n+1)$, где n — размер передаваемого пакета за раз, т. е. длина строящегося циклического кода.

Примеры порождающих полиномов, можно найти в таблице:

r , степень полинома	$P(x)$, порождающий полином
2	111
3	1011
4	10011
5	100101, 111101, 110111
6	1000011, 1100111
7	10001001, 10001111, 10011101
8	111100111, 100011101, 101100011

Алгоритм получения разрешенной кодовой комбинации циклического кода из комбинации простого кода

Пусть задан полином $P(x) = a_{r-1} x^r + a_{r-2} x^{r-1} + \dots + 1$, определяющий корректирующую способность кода и число проверочных разрядов r , а также исходная комбинация простого k -элементного кода в виде многочлена $A_{k-1}(x)$.

Требуется определить разрешенную кодовую комбинацию циклического кода (n, k) .

Умножаем многочлен исходной кодовой комбинации на x^r :

$$A_{k-1}(x) \cdot x^r$$

Определяем проверочные разряды, дополняющие исходную информационную комбинацию до разрешенной, как остаток от деления полученного в предыдущем пункте произведения на порождающий полином:

$$A_{k-1}(x) \cdot x^r / P(x) \Rightarrow (x)$$

Окончательно разрешенная кодовая комбинация циклического кода определится так:

$$A_{n-1}(x) = A_{k-1}(x) \cdot x^r + R(x)$$

Для обнаружения ошибок в принятой кодовой комбинации достаточно поделить ее на производящий полином. Если принятая комбинация — разрешенная, то остаток от деления будет нулевым. Ненулевой остаток свидетельствует о том, что принятая комбинация содержит ошибки. По виду остатка (синдрома) можно в некоторых случаях также сделать вывод о характере ошибки, ее местоположении и исправить ошибку.

Пример. Пусть требуется закодировать комбинацию вида 1101, что соответствует $A(x) = x^3 + x^2 + 1$.

Определяем число контрольных символов $r = 3$. Из таблицы возьмем многочлен $P(x) = x^3 + x + 1$, т. е. 1011.

Умножим $A(x)$ на x^r :

$$A(x) \cdot x^r = (x^3 + x^2 + 1) \cdot x^3 = x^6 + x^5 + x^3 \Rightarrow 11010000$$

Разделим полученное произведение на образующий полином $g(x)$:

$$A(x) \cdot x^r / P(x) = (x^6 + x^5 + x^3) / (x^3 + x + 1) = x^3 + x^2 + x + 1 + 1 / (x^3 + x + 1) \Rightarrow 1111 + 001 / 1011$$

При делении необходимо учитывать, что вычитание производится по модулю 2. Остаток суммируем с $h(x) \cdot x^r$. В результате получим закодированное сообщение:

$$F(x) = (x^3 + x^2 + 1) \cdot (x^3 + x + 1) = (x^3 + x^2 + 1) \cdot x^3 + 1 \Rightarrow 1101001$$

В полученной кодовой комбинации циклического кода информационные символы $A(x) = 1101$, а контрольные $R(x) = 001$. Закодированное сообщение делится на образующий полином без остатка.

Алгоритм определения ошибки

Пусть имеем n -элементные комбинации ($n = k + r$) тогда:

Получаем остаток от деления $E(x)$ соответствующего ошибке в старшем разряде [1000000000], на порождающий полином $Pr(x)$:

$$E1(x) / Pr(x) = R0(x)$$

Делим полученный полином $H(x)$ на $Pr(x)$ и получаем текущий остаток $R(x)$.

Сравниваем $R0(x)$ и $R(x)$.

Если они равны, то ошибка произошла в старшем разряде.

Если нет, то увеличиваем степень принятого полинома на x и снова проводим деления:

$$H(x) \cdot x / Pr(x) = R(x)$$

Опять сравниваем полученный остаток с $R0(x)$.

Если они равны, то ошибки во втором разряде.

Если нет, то умножаем $H(x) \cdot x^2$ и повторяем эти операции до тех пор, пока $R(x)$ не будет равен $R0(x)$.

Ошибка будет в разряде, соответствующем числу, на которое повышена степень $H(x)$, плюс один.

Например: $H(x) \cdot x^3 / \text{Pr}(x) = R0(x)$

Линейные блочные коды

Блочный код определяется, как набор возможных кодов, который получается из последовательности бит, составляющих сообщение. Например, если мы имеем K бит, то имеется 2^K возможных сообщений и такое же число кодов, которые могут быть получены из этих сообщений. Набор этих кодов представляет собой блочный код. Линейные коды получаются в результате перемножения сообщения M на порождающую матрицу $G[IA]$. Каждой порождающей матрице ставится в соответствие матрица проверки четности $(n-k) \cdot n$. Эта матрица позволяет исправлять ошибки в полученных сообщениях путем вычисления синдрома. Матрица проверки четности находится из матрицы идентичности I и транспонированной матрицы A . $G[IA] \Rightarrow H[A^T I]$.

$$IA_A^T$$

$$G[IA] = \begin{bmatrix} 100 & 110 \\ 010 & 011 \\ 001 & 101 \end{bmatrix}, \text{ то } H[A^T I] = \begin{bmatrix} 101 & 100 \\ 110 & 010 \\ 011 & 001 \end{bmatrix}$$

Если

Синдром полученного сообщения равен

$S = [\text{полученное сообщение}] \cdot [\text{матрица проверки четности}]$.

Если синдром содержит нули, ошибок нет, в противном случае сообщение доставлено с ошибкой. Если сообщение M соответствует $M=2^k$, а $k=3$ высота матрицы, то можно записать восемь кодов:

Сообщения	Кодовые вектора	Вычисленные как
$M1 = 000$	$V1 = 000000$	$M1 \cdot G$
$M2 = 001$	$V2 = 001101$	$M2 \cdot G$
$M3 = 010$	$V3 = 010011$	$M3 \cdot G$
$M4 = 100$	$V4 = 100110$	$M4 \cdot G$
$M5 = 011$	$V5 = 011110$	$M5 \cdot G$
$M6 = 101$	$V6 = 101011$	$M6 \cdot G$
$M7 = 110$	$V7 = 110101$	$M7 \cdot G$
$M8 = 111$	$V8 = 111000$	$M8 \cdot G$

Кодовые векторы для этих сообщений приведены во второй колонке. На основе этой информации генерируется таблица 3, которая называется стандартным массивом. Стандартный массив использует кодовые слова и добавляет к ним биты ошибок, чтобы получить неверные кодовые слова. Стандартный массив для кодов (6,3) представлен ниже.

000000	001101	010011	100110	011110	101011	110101	111000
000001	001100	010010	100111	011111	101010	110100	111001
000010	001111	010001	100100	011100	101001	110111	111010
000100	001001	010111	100010	011010	101111	110001	111100
001000	000101	011011	101110	010110	100011	111101	110000
010000	011101	000011	110110	001110	111011	100101	101000
100000	101101	110011	000110	111110	001011	010101	011000
001001	000100	011010	101111	010111	100010	111100	011001

Предположим, что верхняя строка таблицы содержит истинные значения переданных кодов. Из таблицы 3 видно, что, если ошибки случаются в позициях, соответствующих битам кодов из левой колонки, можно определить истинное значение полученного кода. Для этого достаточно полученный код сложить с кодом в левой колонке посредством операции XOR. Синдром равен произведению левой колонки (CL "coset leader") стандартного массива на транспонированную матрицу контроля четности H^T .

Синдром = $CL \cdot H^T$	Левая колонка стандартного массива
000	000000
001	000001
010	000010
100	000100
110	001000
101	010000
011	100000
111	001001

Чтобы преобразовать полученный код в правильный, нужно умножить полученный код на транспонированную матрицу проверки четности, с тем чтобы получить синдром. Полученное значение левой колонки стандартного массива добавляется (XOR!) к полученному коду, чтобы получить его истинное значение. Например, если мы получили 001100, умножаем этот код на H^T :

$$S = \begin{bmatrix} 110 \\ 011 \\ 101 \\ 100 \\ 010 \\ 001 \end{bmatrix} 0011 = 001$$

этот результат указывает на место ошибки, истинное значение кода получается в результате операции XOR:

001100

000001

001101 под горизонтальной чертой записано истинное значение кода.

Варианты заданий:

- 1) Реализовать проверку правильности передачи двоичной кодовой последовательности длиной 8 бит, используя метод кода с одиночным битом четности.
- 2) Реализовать проверку правильности передачи двоичной кодовой последовательности длиной 12 бит, используя метод кода с одиночным битом четности.
- 3) Реализовать проверку правильности передачи двоичной кодовой последовательности, используя метод CRC4 - метод вычисления циклических сумм.
- 4) Реализовать проверку правильности передачи двоичной кодовой последовательности, используя метод CRC8 - метод вычисления циклических сумм.
- 5) Реализовать проверку правильности передачи двоичной кодовой последовательности, используя метод CRC12 - метод вычисления циклических сумм.
- 6) Реализовать проверку правильности передачи двоичной кодовой последовательности, используя метод CRC16 - метод вычисления циклических сумм.
- 7) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод Хэмминга (3,1).
- 8) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод Хэмминга (7,4).
- 9) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод Хэмминга (11,7).
- 10) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод Хэмминга (15,11).
- 11) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод линейных блочных кодов (4,2).
- 12) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод линейных блочных кодов (6,3).

- 13) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод линейных блочных кодов (8,4).
- 14) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод циклических кодов с максимальной степенью полинома 3.
- 15) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод циклических кодов с максимальной степенью полинома 4.
- 16) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод циклических кодов с максимальной степенью полинома 5.
- 17) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод циклических кодов с максимальной степенью полинома 6.
- 18) Реализовать коррекцию ошибки в двоичной кодовой последовательности, используя метод циклических кодов с максимальной степенью полинома 7.

Лабораторная работа №6. Методы сжатия информации

Цель работы: Получить практические навыки по применению различных методов сжатия информации. Получить сравнительную характеристику сжатия информации, используя различные комбинации методов. Сделать вывод.

Теоретические сведения

Сжатие информации - проблема, имеющая достаточно давнюю историю, гораздо более давнюю, нежели история развития вычислительной техники, которая (история) обычно шла параллельно с историей развития проблемы кодирования и шифровки информации.

Все алгоритмы сжатия оперируют входным потоком информации, минимальной единицей которой является бит, а максимальной - несколько бит, байт или несколько байт.

Целью процесса сжатия, как правило, есть получение более компактного выходного потока информационных единиц из некоторого изначально некомпактного входного потока при помощи некоторого их преобразования.

Основными техническими характеристиками процессов сжатия и результатов их работы являются:

- степень сжатия (compress rating) или отношение (ratio) объемов исходного и результирующего потоков;
- скорость сжатия - время, затрачиваемое на сжатие некоторого объема информации входного потока, до получения из него эквивалентного выходного потока;
- качество сжатия - величина, показывающая на сколько сильно упакован выходной поток, при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Все способы сжатия можно разделить на две категории: обратимое и необратимое сжатие.

Под необратимым сжатием подразумевают такое преобразование входного потока данных, при котором выходной поток, основанный на определенном формате информации, представляет, с некоторой точки зрения, достаточно похожий по внешним характеристикам на входной поток объект, однако отличается от него объемом.

Степень сходства входного и выходного потоков определяется степенью соответствия некоторых свойств объекта (т.е. сжатой и несжатой информации в соответствии с некоторым определенным форматом данных), представляемого данным потоком информации.

Такие подходы и алгоритмы используются для сжатия, например данных растровых графических файлов с низкой степенью повторяемости байтов в потоке. При таком подходе используется свойство структуры формата графического файла и возможность представить графическую

картинку приблизительно схожую по качеству отображения (для восприятия человеческим глазом) несколькими (а точнее n) способами. Поэтому, кроме степени или величины сжатия, в таких алгоритмах возникает понятие качества, т.к. исходное изображение в процессе сжатия изменяется, то под качеством можно понимать степень соответствия исходного и результирующего изображения, оцениваемая субъективно, исходя из формата информации. Для графических файлов такое соответствие определяется визуально, хотя имеются и соответствующие интеллектуальные алгоритмы и программы. Необратимое сжатие невозможно применять в областях, в которых необходимо иметь точное соответствие информационной структуры входного и выходного потоков. Данный подход реализован в популярных форматах представления видео и фото информации, известных как JPEG и JFIF алгоритмы и JPG и JIF форматы файлов.

Обратимое сжатие всегда приводит к снижению объема выходного потока информации без изменения его информативности, т.е. - без потери информационной структуры.

Более того, из выходного потока, при помощи восстанавливающего или декомпрессирующего алгоритма, можно получить входной, а процесс восстановления называется декомпрессией или распаковкой и только после процесса распаковки данные пригодны для обработки в соответствии с их внутренним форматом.

Перейдем теперь непосредственно к алгоритмическим особенностям обратимых алгоритмов и рассмотрим важнейшие теоретические подходы к сжатию данных, связанные с реализацией кодирующих систем и способы сжатия информации.

Сжатие способом кодирования серий (RLE)

Наиболее известный простой подход и алгоритм сжатия информации обратимым путем - это кодирование серий последовательностей (Run Length Encoding - RLE).

Суть методов данного подхода состоит в замене цепочек или серий повторяющихся байтов или их последовательностей на один кодирующий байт и счетчик числа их повторений.

Например:

44 44 44 11 11 11 11 01 33 FF 22 22 - исходная последовательность

03 44 04 11 00 03 01 03 FF 02 22 - сжатая последовательность

Первый байт указывает сколько раз нужно повторить следующий байт

Если первый байт равен 00, то затем идет счетчик, показывающий сколько за ним следует неповторяющихся данных.

Данные методы, как правило, достаточно эффективны для сжатия растровых графических изображений (BMP, PCX, TIF, GIF), т.к. последние

содержат достаточно много длинных серий повторяющихся последовательностей байтов.

Недостатком метода RLE является достаточно низкая степень сжатия.

Арифметическое кодирование

Совершенно иное решение предлагает т.н. арифметическое кодирование. Арифметическое кодирование является методом, позволяющим упаковывать символы входного алфавита без потерь при условии, что известно распределение частот этих символов и является наиболее оптимальным, т.к. достигается теоретическая граница степени сжатия.

Предполагаемая требуемая последовательность символов, при сжатии методом арифметического кодирования рассматривается как некоторая двоичная дробь из интервала $[0, 1)$. Результат сжатия представляется как последовательность двоичных цифр из записи этой дроби.

Идея метода состоит в следующем: исходный текст рассматривается как запись этой дроби, где каждый входной символ является «цифрой» с весом, пропорциональным вероятности его появления. Этим объясняется интервал, соответствующий минимальной и максимальной вероятностям появления символа в потоке.

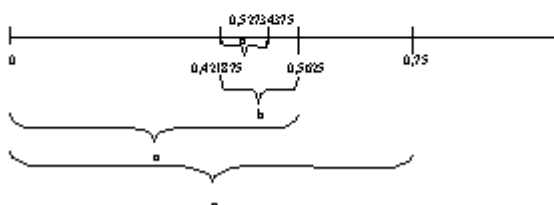
Пример.

Пусть алфавит состоит из двух символов: a и b с вероятностями соответственно $0,75$ и $0,25$.

Рассмотрим наш интервал вероятностей $[0, 1)$. Разобьем его на части, длина которых пропорциональна вероятностям символов. В нашем случае это $[0; 0,75)$ и $[0,75; 1)$. Суть алгоритма в следующем: каждому слову во входном алфавите соответствует некоторый подинтервал из интервала $[0, 1)$ а пустому слову соответствует весь интервал $[0, 1)$. После получения каждого следующего символа интервал уменьшается с выбором той его части, которая соответствует новому символу. Кодом цепочки является интервал, выделенный после обработки всех ее символов, точнее, двоичная запись любой точки из этого интервала, а длина полученного интервала пропорциональна вероятности появления кодируемой цепочки.

Применим данный алгоритм для цепочки "aaba":

Шаг	Предложенная цепочка	Интервал
0	Нет	$[0; 1)$
1	a	$[0; 0,75)$
2	aa	$[0; 0,5625)$
3	aab	$[0,421875; 0,5625)$
4	$aaba$	$[0,421875; 0,52734375)$



Границы интервала вычисляются так берется расстояние внутри интервала ($0,5625 - 0,421875 = 0,140625$), делится на частоты $[0; 0,10546875)$ и $[0,10546875; 1)$ и находятся новые границы $[0,421875; 0,52734375)$ и $[0,52734375; 0,5625)$.

В качестве кода можно взять любое число из интервала, полученного на шаге 4, например, 0,43.

Алгоритм декодирования работает аналогично кодирующему. На входе 0,43 и идет разбиение интервала.

Продолжая этот процесс, мы однозначно декодируем все четыре символа. Для того, чтобы декодирующий алгоритм мог определить конец цепочки, мы можем либо передавать ее длину отдельно, либо добавить к алфавиту дополнительный уникальный символ – «конец цепочки».

Алгоритм Лемпеля-Зива-Велча (Lempel-Ziv-Welch - LZW)

Данный алгоритм отличают высокая скорость работы как при упаковке, так и при распаковке, достаточно скромные требования к памяти и простая аппаратная реализация.

Недостаток - низкая степень сжатия по сравнению со схемой двухступенчатого кодирования.

Предположим, что у нас имеется словарь, хранящий строки текста и содержащий порядка от 2-х до 8-ми тысяч пронумерованных гнезд. Запишем в первые 256 гнезд строки, состоящие из одного символа, номер которого равен номеру гнезда.

Алгоритм просматривает входной поток, разбивая его на подстроки и добавляя новые гнезда в конец словаря. Прочитаем несколько символов в строку s и найдем в словаре строку t - самый длинный префикс s .

Пусть он найден в гнезде с номером n . Выведем число n в выходной поток, переместим указатель входного потока на $\text{length}(t)$ символов вперед и добавим в словарь новое гнездо, содержащее строку $t+c$, где c - очередной символ на входе (сразу после t). Алгоритм преобразует поток символов на входе в поток индексов ячеек словаря на выходе.

При практической реализации этого алгоритма следует учесть, что любое гнездо словаря, кроме самых первых, содержащих одно-символьные цепочки, хранит копию некоторого другого гнезда, к которой в конец приписан один символ. Вследствие этого можно обойтись простой списочной структурой с одной связью.

Пример: ABCABCABCABCABC - 1 2 3 4 6 5 7 7 7

1	A
2	B
3	C
4	AB
5	BC

6	CA
7	ABC
8	CAB
9	BCA

Двухступенчатое кодирование. Алгоритм Лемпеля-Зива

Гораздо большей степени сжатия можно добиться при выделении из входного потока повторяющихся цепочек - блоков, и кодирования ссылок на эти цепочки с построением хеш таблиц от первого до n-го уровня.

Метод, о котором и пойдет речь, принадлежит Лемпелю и Зиву и обычно называется LZ-compression.

Суть его состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые.

Размеры этого буфера, называемого также скользящим словарем (sliding dictionary), варьируются в разных реализациях кодирующих систем.

Экспериментальным путем установлено, что программа LHarc использует 4-килобайтный буфер, LHA и PKZIP - 8-ми, а ARJ - 16-килобайтный.

Затем, после построения хеш таблиц алгоритм выделяет (путем поиска в словаре) самую длинную начальную подстроку входного потока, совпадающую с одной из подстрок в словаре, и выдает на выход пару (length, distance), где length - длина найденной в словаре подстроки, а distance - расстояние от нее до входной подстроки (то есть фактически индекс подстроки в буфере, вычтенный из его размера).

В случае, если такая подстрока не найдена, в выходной поток просто копируется очередной символ входного потока.

В первоначальной версии алгоритма предлагалось использовать простейший поиск по всему словарю. Однако, в дальнейшем, было предложено использовать двоичное дерево и хеширование для быстрого поиска в словаре, что позволило на порядок поднять скорость работы алгоритма.

Таким образом, алгоритм Лемпеля-Зива преобразует один поток исходных символов в два параллельных потока длин и индексов в таблице (length + distance).

Очевидно, что эти потоки являются потоками символов с двумя новыми алфавитами, и к ним можно применить один из упоминавшихся выше методов (RLE, кодирование Хаффмена или арифметическое кодирование).

Так мы приходим к схеме двухступенчатого кодирования - наиболее эффективной из практически используемых в настоящее время. При реализации этого метода необходимо добиться согласованного вывода обоих

потоков в один файл. Эта проблема обычно решается путем поочередной записи кодов символов из обоих потоков.

Пример:

Первая ступень

abcsabcsabcsabc - 1 a 1 b 1 c 3 3 6 3 9 3 12 3

Вторая ступень - исключение большой группы повторяющихся последовательностей

1 a 1 b 1 c 12 3

и сжатие RLE, кодирование Хаффмана, арифметическое кодирование

Алгоритм Хаффмана

Сжимая файл по алгоритму Хаффмана первое что мы должны сделать - это необходимо прочитать файл полностью и подсчитать сколько раз встречается каждый символ из расширенного набора ASCII.

Если мы будем учитывать все 256 символов, то для нас не будет разницы в сжатии текстового и EXE файла.

После подсчета частоты вхождения каждого символа, необходимо просмотреть таблицу кодов ASCII и сформировать бинарное дерево.

Пример:

Мы имеем файл длиной в 100 байт и имеющий 6 различных символов в себе. Мы подсчитали вхождение каждого из символов в файл и получили следующее:

Символ	A	B	C	D	E	F
Число вхождений	10	20	30	5	25	10

Теперь мы берем эти числа и будем называть их частотой вхождения для каждого символа.

Символ	C	E	B	F	A	D
Число вхождений	30	25	20	10	10	5

Мы возьмем из последней таблицы 2 символа с наименьшей частотой. В нашем случае это D (5) и какой либо символ из F или A (10), можно взять любой из них например A.

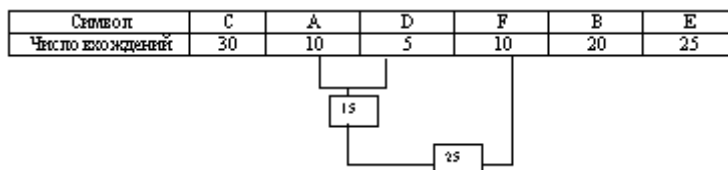
Сформируем из "узлов" D и A новый "узел", частота вхождения для которого будет равна сумме частот D и A:

Символ	C	A	D	F	B	E
Число вхождений	30	10	5	10	20	25

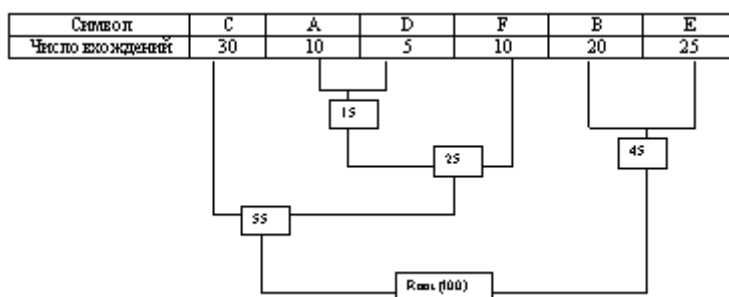
15	-5+10
----	-------

Номер в рамке - сумма частот символов D и A. Теперь мы снова ищем два символа с самыми низкими частотами вхождения. Исключая из просмотра D и A и рассматривая вместо них новый "узел" с суммарной частотой

вхождения. Самая низкая частота теперь у F и нового "узла". Снова сделаем операцию слияния узлов :



Рассматриваем таблицу снова для следующих двух символов (B и E). Мы продолжаем в этот режим пока все "дерево" не сформировано, т.е. пока все не сведется к одному узлу.



Теперь когда наше дерево создано, мы можем кодировать файл . Мы должны всегда начинать из корня (Root). Кодируя первый символ (лист дерева C) Мы прослеживаем вверх по дереву все повороты ветвей и если мы делаем левый поворот, то запоминаем 0-й бит, и аналогично 1-й бит для правого поворота. Так для C, мы будем идти влево к 55 (и запомним 0), затем снова влево (0) к самому символу . Код Хаффмана для нашего символа C - 00. Для следующего символа (A) у нас получается - лево,право,лево,лево , что выливается в последовательность 0100. Выполнив выше сказанное для всех символов получим

C = 00 (2 бита)

A = 0100 (4 бита)

D = 0101 (4 бита)

F = 011 (3 бита)

B = 10 (2 бита)

E = 11 (2 бита)

При кодировании заменяем символы на данные последовательности.

Варианты заданий:

- 1) Выполнить первое сжатие файла способом кодирования серий (RLE), и повторное методом арифметического кодирования.
- 2) Выполнить первое сжатие файла способом кодирования серий (RLE), и повторное, используя алгоритм Лемпеля-Зива-Велча.
- 3) Выполнить первое сжатие файла способом кодирования серий (RLE), и повторное, используя двухступенчатое кодирование. Алгоритм Лемпеля-Зива.
- 4) Выполнить первое сжатие файла способом кодирования серий (RLE), и повторное, используя алгоритм Хаффмана.
- 5) Выполнить первое сжатие файла методом арифметического кодирования, и повторное способом кодирования серий (RLE).
- 6) Выполнить первое сжатие файла методом арифметического кодирования, и повторное, используя алгоритм Лемпеля-Зива-Велча.
- 7) Выполнить первое сжатие файла методом арифметического кодирования, и повторное, используя двухступенчатое кодирование. Алгоритм Лемпеля-Зива.
- 8) Выполнить первое сжатие файла методом арифметического кодирования, и повторное, используя алгоритм Хаффмана.
- 9) Выполнить первое сжатие, используя алгоритм Лемпеля-Зива-Велча, и повторное способом кодирования серий (RLE)
- 10) Выполнить первое сжатие, используя алгоритм Лемпеля-Зива-Велча, и повторное методом арифметического кодирования.
- 11) Выполнить первое сжатие, используя алгоритм Лемпеля-Зива-Велча, и повторное, используя двухступенчатое кодирование. Алгоритм Лемпеля-Зива.
- 12) Выполнить первое сжатие, используя алгоритм Лемпеля-Зива-Велча, и повторное, используя алгоритм Хаффмана.
- 13) Выполнить первое сжатие файла, используя алгоритм Хаффмана, и повторное способом кодирования серий (RLE).

- 14) Выполнить первое сжатие, используя алгоритм Хаффмана, и повторное методом арифметического кодирования.
- 15) Выполнить первое сжатие, используя алгоритм Хаффмана, и повторное, используя алгоритм Лемпеля-Зива-Велча.
- 16) Выполнить первое сжатие, используя алгоритм Хаффмана, и повторное, используя двухступенчатое кодирование. Алгоритм Лемпеля-Зива.
- 17) Выполнить первое сжатие файла, используя двухступенчатое кодирование - алгоритм Лемпеля-Зива, и повторное способом кодирования серий (RLE).
- 18) Выполнить первое сжатие, используя двухступенчатое кодирование - алгоритм Лемпеля-Зива, и повторное методом арифметического кодирования.
- 19) Выполнить первое сжатие, используя двухступенчатое кодирование - алгоритм Лемпеля-Зива, и повторное, используя алгоритм Лемпеля-Зива-Велча.
- 20) Выполнить первое сжатие, используя двухступенчатое кодирование - алгоритм Лемпеля-Зива, и повторное, используя алгоритм Хаффмана.

Лабораторная работа №7. Алгоритмы хеширования паролей

Цель работы: Получить практические навыки по созданию алгоритмов хеширования паролей.

Теоретические сведения

Хеширование паролей

От методов, повышающих криптостойкость системы в целом, перейдем к блоку хеширования паролей – методу, позволяющему пользователям запоминать не 128 байт, то есть 256 шестнадцатиричных цифр ключа, а некоторое осмысленное выражение, слово или последовательность символов, называющуюся паролем. Действительно, при разработке любого криптоалгоритма следует учитывать, что в половине случаев конечным пользователем системы является человек, а не автоматическая система. Это ставит вопрос о том, удобно, и вообще реально ли человеку запомнить 128-битный ключ (32 шестнадцатиричные цифры). На самом деле предел запоминаемости лежит на границе 8-12 подобных символов, а, следовательно, если мы будем заставлять пользователя оперировать именно ключом, тем самым мы практически вынудим его к записи ключа на каком-либо листке бумаги или электронном носителе, например, в текстовом файле. Это, естественно, резко снижает защищенность системы.

Для решения этой проблемы были разработаны методы, преобразующие произносимую, осмысленную строку произвольной длины – пароль, в указанный ключ заранее заданной длины. В подавляющем большинстве случаев для этой операции используются так называемые хеш-функции (от англ. hashing – мелкая нарезка и перемешивание). Хеш-функцией называется такое математическое или алгоритмическое преобразование заданного блока данных, которое обладает следующими свойствами:

1. хеш-функция имеет бесконечную область определения,
2. хеш-функция имеет конечную область значений,
3. она необратима,
4. изменение входного потока информации на один бит меняет около половины всех бит выходного потока, то есть результата хеш-функции.

Эти свойства позволяют подавать на вход хеш-функции пароли, то есть текстовые строки произвольной длины на любом национальном языке и, ограничив область значений функции диапазоном $0..2^N-1$, где N – длина ключа в битах, получать на выходе достаточно равномерно распределенные по области значения блоки информации – ключи.

Нетрудно заметить, что требования, подобные 3 и 4 пунктам требований к хеш-функции, выполняют блочные шифры. Это указывает на один из возможных путей реализации стойких хеш-функций – проведение блочных криптопреобразований над материалом строки-пароля. Этот метод и используется в различных вариациях практически во всех современных

криптосистемах. Материал строки-пароля многократно последовательно используется в качестве ключа для шифрования некоторого заранее известного блока данных – на выходе получается зашифрованный блок информации, однозначно зависящий только от пароля и при этом имеющий достаточно хорошие статистические характеристики. Такой блок или несколько таких блоков и используются в качестве ключа для дальнейших криптопреобразований.

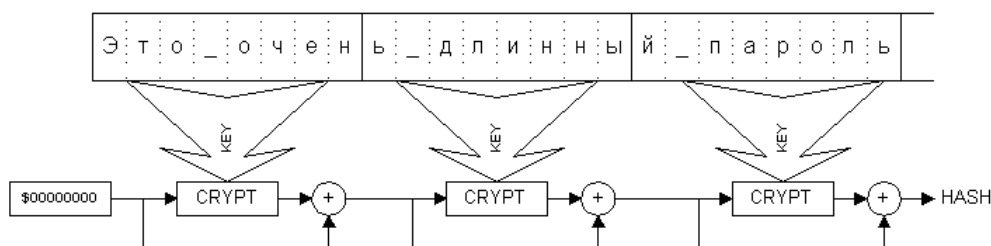
Характер применения блочного шифра для хеширования определяется отношением размера блока используемого криптоалгоритма и разрядности требуемого хеш-результата.

Если указанные выше величины совпадают, то используется схема одноцепочечного блочного шифрования. Первоначальное значение хеш-результата H_0 устанавливается равным 0, вся строка-пароль разбивается на блоки байт, равные по длине ключу используемого для хеширования блочного шифра, затем производятся преобразования по рекуррентной формуле:

$$H_j = H_{j-1} \text{ XOR } \text{EnCrypt}(H_{j-1}, \text{PSW}_j),$$

где $\text{EnCrypt}(X, \text{Key})$ – используемый блочный шифр.

Последнее значение H_k используется в качестве искомого результата.



В том случае, когда длина ключа ровно в два раза превосходит длину блока, а подобная зависимость довольно часто встречается в блочных шифрах, используется схема, напоминающая сеть Фейштеля. Характерным недостатком и приведенной выше формулы, и хеш-функции, основанной на сети Фейштеля, является большая ресурсоемкость в отношении пароля. Для проведения только одного преобразования, например, блочным шифром с ключом длиной 128 бит используется 16 байт строки-пароля, а сама длина пароля редко превышает 32 символа. Следовательно, при вычислении хеш-функции над паролем будут произведено максимум 2 «полноценных» криптопреобразования.

Решение этой проблемы можно достичь двумя путями: 1) предварительно «размножить» строку-пароль, например, записав ее многократно последовательно до достижения длины, скажем, в 256 символов; 2) модифицировать схему использования криптоалгоритма так, чтобы материал строки-пароля "медленнее" тратился при вычислении ключа.

По второму пути пошли исследователи Девис и Майер, предложившие алгоритм также на основе блочного шифра, но использующий матери-

ал строки-пароля многократно и небольшими порциями. В нем просматриваются элементы обеих приведенных выше схем, но криптостойкость этого алгоритма подтверждена многочисленными реализациями в различных криптосистемах. Алгоритм получил название «Tandem DM»:

$G_0=0; H_0=0$;

FOR J = 1 TO N DO

BEGIN

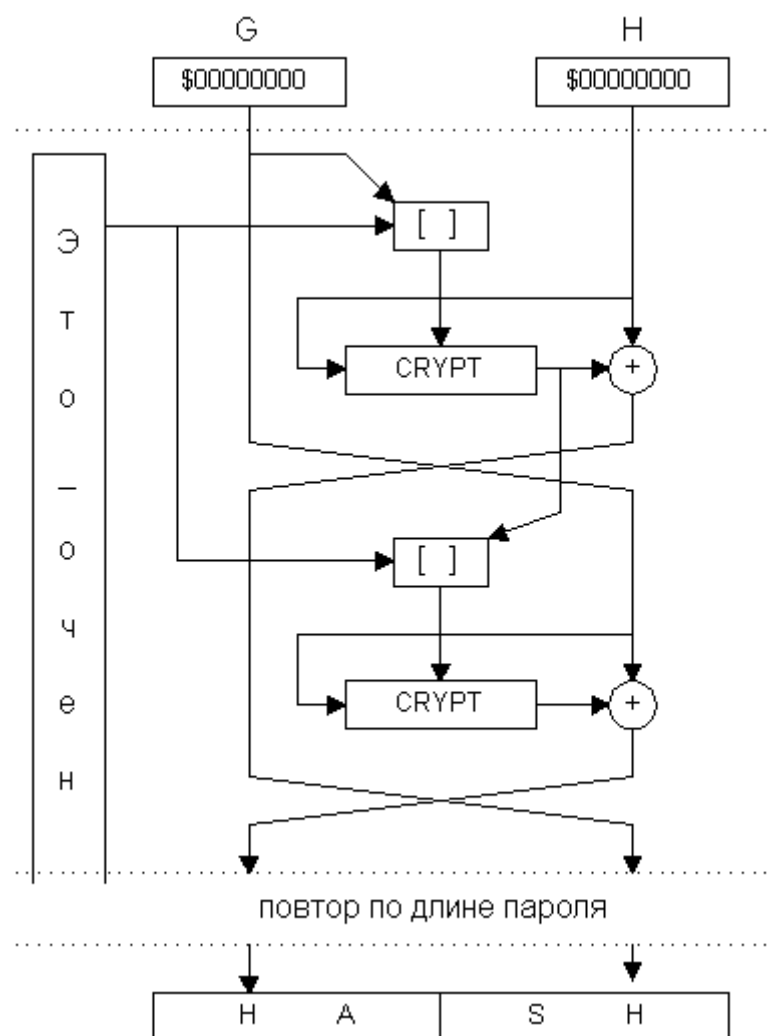
TMP=EnCrypt(H,[G,PSW_j]); H'=H XOR TMP;

TMP=EnCrypt(G,[PSW_j,TMP]); G'=G XOR TMP;

END;

Key=[G_k,H_k]

Квадратными скобками (X16=[A8,B8]) здесь обозначено простое объединение (склеивание) двух блоков информации равной величины в один – удвоенной разрядности. А в качестве процедуры EnCrypt(X,Key) опять может быть выбран любой стойкий блочный шифр. Как видно из формул, данный алгоритм ориентирован на то, что длина ключа двукратно превышает размер блока криптоалгоритма. А характерной особенностью схемы является тот факт, что строка пароля считывается блоками по половине длины ключа, и каждый блок используется в создании хеш-результата дважды. Таким образом, при длине пароля в 20 символов и необходимости создания 128 битного ключа внутренний цикл хеш-функции повторится 3 раза.



Варианты заданий:

Написать программу, реализующую методику хеширования паролей, используя в качестве блочного шифра для реализации алгоритма написанный ранее в лабораторной работе №4 блочный шифр. Максимальная длина пароля выбирается разработчиком алгоритма на его усмотрение, но не должна быть меньше 4 символов.

Лабораторная работа №8. Методы гаммирования

Цель работы: Получить практические навыки по применению метода шифрования однократной случайной равновероятной гаммой – однократного гаммирования.

Теоретические сведения

Шифрование методом гаммирования

Под гаммированием понимают процесс наложения по определенному закону гаммы шифра на открытые данные. Гамма шифра - это псевдослучайная последовательность, выработанная по заданному алгоритму для зашифрования открытых данных и расшифрования зашифрованных данных.

Процесс зашифрования заключается в генерации гаммы шифра и наложении полученной гаммы на исходный открытый текст обратимым образом, например с использованием операции сложения по модулю 2. Следует отметить, что перед зашифрованием открытые данные разбивают на блоки T_0^i одинаковой длины, обычно по 64 бита. Гамма шифра вырабатывается в виде последовательности блоков $\Gamma_{ш}^i$ аналогичной длины.

Уравнение зашифрования можно записать в виде

$$T_{ш}^i = \Gamma_{ш}^i \oplus T_0^i, i = 1...M$$

где $T_{ш}^i$ - i -й блок шифртекста; $\Gamma_{ш}^i$ - i -й блок гаммы шифра; T_0^i - i -й блок открытого текста; M - количество блоков открытого текста.

Процесс расшифрования сводится к повторной генерации гаммы шифра и наложению этой гаммы на зашифрованные данные. Уравнение расшифрования имеет вид обратный вид.

Получаемый этим методом шифртекст достаточно труден для раскрытия, поскольку теперь ключ является переменным. По сути дела гамма шифра должна изменяться случайным образом для каждого шифруемого блока. Если период гаммы превышает длину всего шифруемого текста и злоумышленнику неизвестна никакая часть исходного текста, то такой шифр можно раскрыть только прямым перебором всех вариантов ключа. В этом случае криптостойкость шифра определяется длиной ключа.

«Разоблачение» гаммирования.

С точки зрения теории криптоанализа метод шифрования однократной случайной равновероятной гаммой той же длины, что и открытый текст, является невскрываемым (далее для краткости авторы будут употреблять термин «однократное гаммирование», держа в уме все вышесказанное). Обоснование, которое привел Шеннон, основываясь на введенном им же понятии информации, не дает возможности усомниться в этом - из-за равных априорных вероятностей криптоаналитик не может сказать о дешифровке, верна она или нет. Кроме того, даже раскрыв часть сообще-

ния, дешифровщик не сможет хоть сколько-нибудь поправить положение - информация о вскрытом участке гаммы не дает информации об остальных ее частях.

Логично было бы предположить, что для организации канала конфиденциальной связи в открытых сетях следовало бы воспользоваться именно схемой шифрования однократного гаммирования. Ее преимущества вроде бы очевидны. Есть, правда, один весомый недостаток, который сразу бросается в глаза, - это необходимость иметь огромные объемы данных, которые можно было бы использовать в качестве гаммы. Для этих целей обычно пользуются датчиками настоящих случайных чисел (в западной литературе аналогичный термин носит название True Random Number Generator или TRNG). Это уже аппаратные устройства, которые по запросу выдают набор случайных чисел, генерируя их с помощью очень большого количества физических параметров окружающей среды. Статистические характеристики таких наборов весьма близки к характеристикам "белого шума", что означает равновероятное появление каждого следующего числа в наборе. А это, в свою очередь, означает для нас действительно равновероятную гамму.

К сожалению, для того чтобы организовать конфиденциальный канал передачи данных, потребуется записать довольно большое количество этих данных и обмениваться ими по секретному каналу. Уже одно это условие делает однократное гаммирование во многих случаях неприемлемым. В самом деле, зачем передавать что-то по открытому незащищенному каналу, когда есть возможность передать все это по секретному защищенному? И хотя на простой вопрос, является ли метод использования однократной случайной равновероятной гаммы стойким к взлому, существует положительный ответ, его использование может оказаться попросту невозможным.

Да и к тому же метод однократного гаммирования криптостоек только в определенных, можно даже сказать, тепличных условиях. Что же касается общего случая, то все не так просто.

Показать слабости шифра однократного гаммирования можно, говоря научно, с помощью примера или, что называется, "на пальцах". Представим следующую ситуацию.

Допустим, в тайной деловой переписке используется метод однократного наложения гаммы на открытый текст.

Чтобы дальнейшие рассуждения были как можно более понятны, рассмотрим следующее свойство шифротекста. Предположим, что мы знаем часть гаммы, которая была использована для шифрования текста «Приветствую, мой ненаглядный сосед!» в формате ASCII, кодировка WIN-1251.

Приветствую, _ мой
CF F0 E8 E2 E5 F2 F1 F2 E2 F3 FE 2C 20 EC EE E9

— н е н а г л я д н ы й — с о с
20 ED E5 ED E0 E3 EB FF E4 ED FB E9 20 F1 EE F1
е д !
E5 E4 21

Варианты заданий:

- 1) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию сложения.
- 2) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию исключающее или.
- 3) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию умножение по модулю 2^N , где за N взять число блоков открытого текста.
- 4) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 48 бита и используя в алгоритме шифрования операцию сложения.
- 5) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 48 бита и используя в алгоритме шифрования операцию исключающее или.
- 6) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 48 бита и используя в алгоритме шифрования операцию умножение по модулю 2^N , где за N взять число блоков открытого текста.
- 7) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 64 бита и используя в алгоритме шифрования операцию сложения.
- 8) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 64 бита и используя в алгоритме шифрования операцию исключающее или.
- 9) Реализовать шифрование сообщения методом однократного гаммирования, используя блоки открытого текста длиной 64 бита и используя в алгоритме шифрования операцию умножение по модулю 2^N , где за N взять число блоков открытого текста.
- 10) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию сложения.

- 11) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию исключающее или.
- 12) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 32 бита и используя в алгоритме шифрования операцию умножение по модулю 2^N , где за N взять число блоков открытого текста.
- 13) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 48 бита и используя в алгоритме шифрования операцию сложения.
- 14) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 48 бита и используя в алгоритме шифрования операцию исключающее или.
- 15) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 48 бита и используя в алгоритме шифрования операцию умножение по модулю 2^N , где за N взять число блоков открытого текста.
- 16) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 64 бита и используя в алгоритме шифрования операцию сложения.
- 17) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 64 бита и используя в алгоритме шифрования операцию исключающее или.
- 18) Реализовать шифрование файла методом однократного гаммирования, используя блоки открытого текста длиной 64 бита и используя в алгоритме шифрования операцию умножение по модулю 2^N , где за N взять число блоков открытого текста.

Список литературы

1. Аршинов М.Н., Садовский Л.Е. Коды и математика (рассказы о кодировании). – М.: Наука, 1983. – 144с.
2. Введение в криптографию / Под общ. ред. В.В. Ященко. Серия: Учебники для технических ВУЗов. – СПб.: Питер, 2001. – 288с.
3. Герасименко В.А., Скворцов А.А., Харитонов И.Е. Новые направления применения криптографических методов защиты информации.- М.: Радио и связь, 1989. – 360 с.
4. Мельников В. В. Защита информации в компьютерных системах. – М.: Финансы и статистика; Электронинформ, 1997. – 368 с.
5. Нечаев В.И. Элементы криптографии (основы теории защиты информации): учебное пособие для университетов и педвузов./Под ред. В.А. Садовничева. - М.: Высш. шк., 1999. – 109с.
6. Спесивцев А.В. Защита информации в персональных ЭВМ. – М.: Радио и связь, 1992. – 190 с.
7. Столлингс В. Криптография и защита сетей: принципы и практика. Пер. с англ. – М.: Вильямс, 2001. – 672 с.
8. Фергюсон Н., Шнайер Б., Практическая криптография. – М: Вильямс, 2005. – 242с.
9. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.: Триумф, 2002. – 816с.