МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

РОБОЧА ПРОГРАМА,
методичні вказівки та індивідуальні завдання до вивчення дисципліни «Технології захисту інформації» для студентів спеціальності 122-комп'ютерні науки

Дніпро НМетАУ 2019

УДК 681.3.06+519.68

Методичні вказівки з дисципліні «Технології захисту інформації». Для студентів напряму 0501 01 – «Комп'ютерні науки» / Укл.: О.І. Деревянко– Дніпро: НМетАУ, 2019. – 27 с.

Методичні вказівки є практичною частиною комплексу навчально-методичних матеріалів з дисципліні «Технології захисту інформації», в якій розглянуті загальні принципи і математичні моделі систем захисту інформації, базові алгоритми та методи програмної їх реалізації.

Методичні вказівки призначені для студентів напряму підготовки 122 – «Комп'ютерні науки», а також для слухачів курсів підвищення кваліфікації, студентів і аспірантів інших спеціальностей.

Укладачі: О.І. Деревянко, кандидат технічних наук, професо,

Відповідальний за випуск: О.І. Михальов, д-р техн. наук, проф.

Рецензент: В.І. Корсун, доктор технічних наук, професор

Друкується за авторською редакцією.

Затверджено на засіданні кафедри інформаційних технологій і систем, протокол № 9 від 06.03.2019.

Підписано до друку 10.05.2019. Формат 60х84 1/16. Папір типогр.

Друк різограф. Облік.-вид. арк 4,75. Умов. друк. арк. 3,40.

Тираж 100 пр. Замовл. № 19/12.

ДНВП «Системні технології»

Шифрование методами замены

В криптографии рассматриваются четыре типа подстановки (замены): моноалфавитная, полиалфавитная, гомофоническая и полиграммная.

Далее всюду в примерах использовано кодирование букв русского алфавита, приведенное в табл. 1.

Таблица 1. Кодирование букв русского алфавита

Буква	А	Б	В	Г	Д	Е	ж	З	И	И	К	Л
Код	00	01	02	03	04	05	06	07	08	09	10	11
Буква	М	Н	о	П	Р	С	Т	У	Ф	Х	Ц	Ч
Код	12	13	14	15	16	17	18	19	20	21	22	23
Буква	Ш	щ	Ъ	ы	Ь	Э	Ю	Я	D (пробел)			
Код	24	25	26	27	28	29	30	31	32			

При ***моноалфавитной замене*** каждой букве алфавита открытого текста ставится в соответствие одна буква шифртекста из этого же алфавита.

Общая формула моноалфавитной замены выглядит следующим образом

$$y_i = k_1 x_i + k_2\text{-}(mod n),$$

где y_i — i-й символ алфавита k_1 и k_2 — константы, x_i — i-й символ открытого текста (номер буквы в алфавите), n — длина используемого алфавита

Основным недостатком рассмотренного метода является то, что статистические свойства открытого текста (частоты появления букв) сохраняются и в шифртексте

Пример 1. Открытый текст «ШИФРОВАНИЕ ЗАМЕНОЙ» Подстановка задана табл 2

Таблица 2 Подстановка алфавита для шифрования заменой

Алфавит исходного текста	А	Б	В	Г	Д	…	Ь	Э	Ю	Я	А
Алфавит шифртекста	D	Я	Ю	Э	Ь	…	Д	Г	В	Б	А

Шифртекст «ИШМРТЮПУШЫАЩаФЫУТЧ»

Шифр, задаваемый формулой

$$y_i = x_i + ki\text{-}(mod n),$$

где *ki* — i- *я* буква ключа, в качестве которого используется слово или фраза называется *шифром Вижинера*

Пример 2. Открытый текст «ЗАМЕНА» Подстановка задана в табл 3

Таблица 3 Подстановка шифра Вижинера

З	А	М	Е	н	А

К	Л	Ю	Ч	к	Л

$y_i = 8 + 11(\text{mod}33) = 19 \rightarrow$- Т,

уз = 1 + 12(mod33) = 13 → *M,*

$y_i = 13+31(\text{mod}33)= 11 \rightarrow$ А,

У4 = 6 + 24(mod 33) = 30 → Э,

$y_i = 14 + 11(\text{mod}33) = 25 \rightarrow$ Ш,

ув=1+ 12(mod 33) = 13 → *M*

Шифртекст «ТМКЭШМ»

Шифры Бофора используют формулы

$$y, = k, - x, (\text{mod n}) \text{ и } y = \text{ж}, — \text{fci}(\text{modn})$$

Полиалфавитная замена использует несколько алфавитов шифртекста Пусть используется *k* алфавитов Тогда открытый текст

$$X = XiX\text{-}l...XkXk+\backslash\text{---}X2kX\text{-}2k+\backslash ...$$

заменяется шифртекстом

$$Y = /\text{l}(\text{^l})/2(\text{^2}) . . .\text{A}(\text{^fc})/\text{l}(\text{a-t+l}) . . \bullet fk\{x'ik\}fl(x2k+l) \bullet\bullet\text{-},$$

где *fi[x^)* означает символ шифртекста алфавита г для символа открытого текста *Xj.*

Пример 3. Открытый текст: «ЗАМЕНА», *k* = 3. Подстановка задана таблицей 4 Шифртекст- «7б 31 61 97 84 48».

Гомофоническая замена одному символу открытого текста ставит в соответствие несколько символов шифртекста. Этот метод применяется для искажения статистических свойств шифртекста.

Пример 4. Открытый текст «ЗАМЕНА» Подстановка задана табл. 4.

Таблица .4- Подстановка алфавита гомофонической замены

Алфавит открытого	А	Б	…	Е	Ж	З	…	М	Н	…
Алфавит шифртекста	17 31 48	23 44 63	… …	97 51 15	47 67 33	76 19 59	… …	32 28 61	55 84 34	… …

Шифртекст. «76 17 32 97 55 31»

Таким образом, при гомофонической замене каждая буква открытого текста заменяется по очереди цифрами соответствующего столбца.

Полиграммная замена формируется из одного алфавита с помощью специальных правил. В качестве примера рассмотрим шифр *Плэйфера*

В этом шифре алфавит располагается в матрице Открытый текст разбивается на пары символов *x^.x,^.* Каждая пара символов открытого текста заменяется на

пару символов из матрицы следующим образом:

- если символы находятся в одной строке, то каждый из символов пары заменяется на стоящий правее его (за последним символом в строке следует первый),
- если символы находятся в одном столбце, то каждый символ пары заменяется на символ, расположенный ниже его в столбце (за последним нижним символом следует верхний),
- если символы пары находятся в разных строках и столбцах, то они считаются противоположными углами прямоугольника Символ, находящийся в левом углу, заменяется на символ, стоящий в другом левом углу, замена символа, находящегося в правом углу, осуществляется аналогично,
- если в открытом тексте встречаются два одинаковых символа подряд, то перед шифрованием между ними вставляется специальный символ (например тире)

Пример 5. Открытый текст. «ШИФР ПЛЭЙФЕРА» Матрица алфавита представлена в табл. 5.

Таблица.5. Матрица алфавита шифра Плэйфера

А	Ж	Б	м	ц	в
Ч	Г	Н	ш	д	о
Е	Щ	.	х	У	п
	З	ъ	р	и	и
С	ь	к	э	т	л
Ю	я	а	ы	ф	—

Шифртекст: «РДЫИ-СТ-И.ХЧС»

При рассмотрении этих видов шифров становится очевидным, что чем больше длина ключа (например в шифре Вижинера), тем лучше шифр Существенного улучшения свойств шифртекста можно достигнуть при использовании шифров с автоключом

Шифр, в котором сам открытый текст или получающаяся криптограмма используются в качестве ключа, называется шифром с автоключом. Шифрование в этом случае начинается с ключа, называемого первичным, и продолжается с помощью открытого текста или криптограммы, смещенной на длину первичного ключа

Пример 6. Открытый текст «ШИФРОВАНИЕ ЗАМЕНОЙ». Первичный ключ. «КЛЮЧ». Схема шифрования с автоключом при использовании открытого текста представлена в табл. 6.

Таблица.6. Схема шифрования с автоключом при использовании открытого текста

Ш	и	Ф	Р	0	в	А	Н	и	Е	а	З	А	м	Е	Н	0	И
К	л	Ю	Ч	Ш	и	Ф	Р	о	В	А	Н	И	Е	а	З	А	М
36	21	52	41	40	12	22	31	24	09	34	22	10	19	39	22	16	23
в	Ф	Т	З	Ж	Л	Х	Ю	Ч	И	А	Х	И	Т	Е	Х	П	Ц

Схема шифрования с автоключом при использовании криптограммы представлена в табл. 7.

Таблица .7. Схема **шифрования с автоключом при использовании** криптограммы

Ш	и	Ф	Р	о	В	А	Н	И	Е	а	З	А	М	Е	Н	0	И
К	л	Ю	Ч	в	Ф	Т	З	С	Ч	У	Х	Ъ	Э	У	Э	Ы	И

36	21	52	41	18	24	20	22	27	30	53	30	24	43	26	44	39	20
в	Ф	Т	З	С	Ч	У	Х	Ъ	э	У	э	Ы	И	Щ	К	И	У

Для шифрования используются и другие методы перестановки символов открытого текста в соответствии с некоторыми правилами

Пример 7. Открытый текст «ШИФРОВАНИЕ ПЕРЕСТАНОВКОЙ»

Ключ (правило перестановки) группы из восьми букв с порядковыми номерами 1 2 8 переставить в порядок 38152764

Шифртекст «ФНШОИАВРПСИЕЕЕРПНИТВАОКО»

Можно использовать и усложненную перестановку Для этого открытый текст записывается в матрицу по определенному ключу А] Шифртекст образуется при считывании из этой матрицы по ключу *Ку*

Пример .8. Открытый текст «ШИФРОВАНИЕ ПЕРЕСТАНОВКОЙ»

Матрица из четырех столбцов приведена в табл 8, где запись по строкам в соответствии с ключом /\i 5 3 1 2 4 6. а чтение по столбцам в соответствии с ключом Кз 4231

Таблица 8. Матрица алфавита с перестановкой из четырех столбцов

1	И	Е	D	п
2	Е	Р	Е	с
3	0	В	А	Н
4	Т	А	Н	о
5	Ш	И	Ф	Р
6	в	К	0	и
к>^2	1	2	3	4

Шифртекст «ПСНОРЙЕРВАИКПЕАНФОИЕОТШВ»

Наиболее сложные перестановки осуществляются по гамильтоновым путям, которых в графе может быть несколько

Пример 9. Открытый текст «ШИФРОВАНИЕ ПЕРЕСТАНОВКОЙ» Ключ — гамильтонов путь на графе (рис 2)

Шифртекст «ШАОНИРФВИЕЕСЕППРТОВЙАОНК»

Чтение криптограммы (1-7-5-8-2-4-3-6)

Запись открытого текста (1-2-3-4-5-6-7-8)

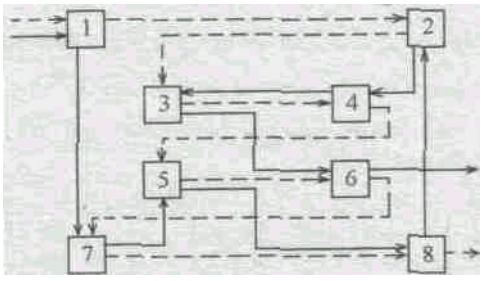


Рис. 2. Гамильтонов путь на графе

Необходимо отметить, что для данного графа из восьми вершин можно предложить несколько маршрутов записи открытого текста и несколько гамильтоновых путей для чтения криптограмм.

В 1991 г В М Кузьмин предложил схему перестановки, основанную на кубике Рубика Согласно этой схеме открытый текст записывается в ячейки граней куба по строкам После осуществления заданного числа заданных поворотов слоев куба считывание шифртекста осуществляется по столбцам Сложность расшифрования в этом случае определяется числом ячеек на гранях куба и сложностью выполненных поворотов слоев. Перестановка, основанная на кубике Рубика, получила название объемной (многомерной) перестановки

В 1992—94 гг идея применения объемной перестановки для шифрования открытого текста получила дальнейшее развитие Усовершенствованная схема перестановок по принципу кубика Рубика, в которой наряду с открытым текстом перестановке подвергаются и функциональные элементы самого алгоритма шифрования, легла в основу секретной системы «Рубикон» В качестве прообразов пространственных многомерных структур, на основании объемных преобразований которых осуществляются перестановки, в системе «Рубикон» используются трехмерный

Шифрование с помощью датчика псевдослучайных чисел

Принцип шифрования заключается в генерации гаммы шифра с помощью датчика псевдослучайных чисел (ПСЧ) и наложении полученной гаммы на открытые данные обратимым образом (например, при использовании логической операции «исключающее ИЛИ»).

Процесс дешифрования данных сводится к повторной генерации гаммы шифра при известном ключе и наложению такой гаммы на зашифрованные данные. Полученный зашифрованный текст является достаточно трудным для раскрытия в том случае, когда гамма шифра не содержит повторяющихся битовых последовательностей. По сути дела гамма шифра должна изменяться случайным образом для каждого шифруемого слова. Фактически если период гаммы превышает длину всего зашифрованного текста и неизвестна никакая часть исходного текста, то шифр можно раскрыть только прямым перебором (подбором ключа). В этом случае криптостойкость определяется размером ключа.

Чтобы получить линейные последовательности элементов гаммы, длина которых превышает размер шифруемых данных, используются датчики ПСЧ. На основе теории групп было разработано несколько типов таких датчиков.

В настоящее время наиболее доступными и эффективными являются конгруэнтные генераторы ПСЧ. Для этого класса генераторов ПСЧ можно сделать математически строгое заключение о том, какими свойствами обладают выходные сигналы этих генераторов с точки зрения периодичности и случайности.

Одним из хороших конгруэнтных генераторов является линейный конгруэнтный датчик ПСЧ. Он вырабатывает последовательности псевдослучайных чисел T(i), описываемые соотношением

$$T(i+1) = (A*T(i)+C) mod M,$$

где A и C — константы, T(0) — исходная величина, выбранная в качестве порождающего числа.

Такой датчик ПСЧ генерирует псевдослучайные числа с определенным периодом повторения, зависящим от выбранных значений A и C. Значение M обычно устанавливается равным 2^b, где b - длина слова компьютера в битах**.** Датчик имеет максимальный период M до того, как генерируемая последовательность чисел начнет повторяться. По причине, отмеченной ранее, необходимо выбирать числа *A* и C таким образом, чтобы период M был максимальным. Как показано в Д. Кнут Искусство программирования для ЭВМ.— М.: Мир, 1976. — Т.2, линейный конгруэнтный датчик ПСЧ имее у т максимальную длину M тогда и только тогда, когда C — нечетное, и (A)mod4=1.

Выше было сказано, что при определенных условиях криптостойкость растет с увеличением размера ключа. Для шифрования данных с помощью датчика ПСЧ может быть выбран ключ любого размера. Например, пусть ключ состоит из набора чисел X(j) размерностью b*,* где /=1,2, ..., N*.* Тогда создаваемую гамму шифра G можно представить как объединение непересекающихся множеств H(j):

$$G = H(1) U H(2) U ... U H(N),$$

где H(j*)* — множество соответствующих j-му сегменту данных и полученных на

основе порождающего числа Y(j), определенного как функция от X(j) (например, ПСЧ, полученное на основе X(j)).

Разумеется, возможны и другие, более изощренные варианты выбора порождающих чисел для гаммы шифра. Более того, гамму шифра необязательно рассматривать как объединение непересекающихся множеств. Например, гамма шифра может быть представлена в следующем виде:

$$G = L(1) (+) L(2) (+) ... (+) L(.N).$$

Здесь символ (+) обозначает операцию «Исключающее ИЛИ», а множества L(j), для каждого из которых мощность равна мощности гаммы, представляют собой объединение следующих множеств:

$$L(J) = V(j)UH(j)UW(j),$$

где V(j) и W(j) - множества нулей, H(j) - множество ПСЧ, соответствующих j-сегменту данных. Причем мощности всех трех множеств выбраны на основе ключа, исходя из того, что мощность L(j) равна мощности G.

Пример простейшей программы шифрования области памяти методом гаммирования с использованием датчика псевдослучайных чисел приведен в приложении

Шифрование с помощью датчика ПСЧ является довольно распространенным криптографическим методом. Во многом качество шифра, построенного на основе датчика ПСЧ, определяется не только и не столько характеристиками датчика, сколько алгоритмом получения гаммы. Один из фундаментальных принципов криптологической практики гласит: даже очень грозно выглядящие шифры могут быть чувствительны к простым воздействиям. Кроме этого, шифры могут быть легко раскрыты, когда не применяются меры предосторожности. В качестве иллюстрации данного принципа рассмотрим проблему известного исходного текста.

Перспективный с практической точки зрения шаг на пути раскрытия любого зашифрованного файла — получить часть некоторого исходного текста и соответствующую ему часть зашифрованного. Общеизвестно, что стандартная информация, например, гриф «СОВ. СЕКРЕТНО» часто является уязвимой. Предположим, возможность добавлять записи к файлу и проверять зашифрованный файл, до и после добавления известной записи. Если гамма шифра представляет собой последовательность псевдослучайных чисел, каждое из которых может быть сгенерировано из предыдущего, то весь исходный текст можно легко восстановить из зашифрованного текста. Рассмотрим последовательность P = p(1),..., p(n) из n исходных слов в файле, к которым после y-го слова, $1<y<n$, добавляется новый элемент текста, содержащий w слов. В результате получается обновленный текст

$$p' = p'(1),..,, p'(n+w).$$

Очевидно, что

$p'(i) = p'(i),\ i = 1, 2, ..., y,$

$p(i) = p'(i+w),..., i = y+1, y+2, .., n.$

Здесь p'(y+l), ... ,p'(y+w) являются известными словами исходного текста.

Пусть G = g(l), g(2) ...— последовательность слов гаммы шифра, используемых для шифрования как P, так и P′. Тогда зашифрованные тексты для P и P′ можно представить в виде

$C = c'(1), ... , c'(n),$

где $c(i) = p(i) (+) gf(i), i= 1, 2, ... л;$

$C' = c'(1),..., c'(n),$

где $c'(i) = p'(i) (+) g'(i),\ i = 1, 2, ..., n+w.$

Теперь можно вычислить слово гаммы, которое использовалось для закрытия известного исходного текста:

$$g(y+1)= p'(y+i) (+) c'(y+i),\ i = 1 ,2 ,..., w.$$

Но эти слова гаммы использовались для шифрования P. Следовательно,

$$p(y+1)= g(y+i) (+) c(y+i), \text{ i} = 1, 2,..., w.$$

Видно, что дешифрование можно повторить, подставив y+w вместо *y.* Таким образом, все сегменты текста после позиции y могут быть дешифрованы. Легкое дешифрование текста стало возможным в связи с тем, что алгоритм шифрования не зависит ни от длины шифруемого файла, ни от содержимого самого файла. Но более или менее серьезное усовершенствование алгоритма получения гаммы шифра приводит к существенному повышению криптостойкости. Хорошие результаты дает метод гаммирования с обратной связью, который заключается в том, что для получения сегмента гаммы используется контрольная сумма определенного участка шифруемых данных. Например, если рассматривать гамму шифра как объединение непересекающихся множеств H(j), то процесс шифрования данных можно представить следующими шагами:

- определение контрольной суммы участка данных, соответствующего сегменту гаммы H(1);
- генерация сегмента гаммы H(1) и наложение его на соответствующий участок шифруемых данных;
- генерация с учетом контрольной суммы уже зашифрованного участка данных следующего сегмента гаммы H(2) (обычно контрольная сумма используется в процессе генерации порождающего числа для очередного сегмента гаммы);
- подсчет контрольной суммы участка данных, соответствующего сегменту гаммы H(2), и наложение этого сегмента гаммы на соответствующий участок шифруемых данных и т. д.

Под контрольной суммой здесь понимается функция f(t(1*), ...,* t(n)), где t(i) - i-e слово шифруемых данных. Разумеется, метод гаммирования с обратной связью может быть реализован с помощью другого алгоритма. Здесь изложены только

общие принципы метода обратной связи (использование некоторых характеристик шифруемых данных для генерации гаммы).

ЗАДАНИЕ.

1. Разработать процедуру побайтного шифрования-дешифрования данных в оперативной памяти компьютера.
2. Исследовать характеристики датчика ПСЧ.
3. Провести анализ работы разработанной программы.
4. Приложение
5.
6. ПРОГРАММА ШИФРОВАНИЯ. ОБЛАСТИ ПАМЯТИ С ПОМОЩЬЮ ПРОГРАММНОГО ДАТЧИКА ПСЧ
7. ***Функции.***
8. Область памяти, указанная вызывающей программой, шифруется по специальному алгоритму (с использованием датчика ПСЧ). Поскольку алгоритм шифрования является обратимым (содержимое области памяти складывается с гаммой шифра по команде "Исключающее ИЛИ"), дешифрование закодированной области памяти заключается в его повторном шифровании (повторном подключении данной процедуры с теми же параметрами).
9. ***ВХОДНЫЕ ПАРАМЕТРЫ***.
10. Все параметры шифрования и характеристики кодируемой области памяти передаются на следующих регистрах: регистры ES:DI указывают на кодируемую область, регистр CX содержит длину этой области; регистр BP содержит коэффициент A для получения очередного ПСЧ, используемого для шифрования области, регистр SI содержит коэффициент C, регистр BX -начальное значение для запуска датчика ПСЧ.
11. ***ВЫХОДНЫЕ ПАРАМЕТРЫ.***
12. После выполнения процедуры регистры ES:DI будут указывать на байт, следующий за обработанной областью, регистр CX будет обнулен, а в регистр BX загрухено ПСЧ. Остальные регистры не изменяются.
13.
14. **push ax**; сохранить рабочие регистры.
15. **push dx**
16. **cme_l: mov ax,bx**; вычислить очередной коэффициент случайного числа.
17. **mul bp**
18. **add ax, si**
19. **mov bx,ax**; сохранить очередной коэффициент в регистре BX.
20. **xor BYTE PTR es:[di],a1** ; закодировать один байт области.
21. **inc di**; перейти к следующему байту кодируемой области.
22. **loop cme_l**
23. **pop dx**; восстановить рабочие регистры.
24. **pop ax**
25. **ret**

Стандарт шифрования данных DES (Data Encryption Standard)

Введение

Стандарт шифрования данных DES (Data Encryption Standard), который ANSI называет Алгоритмом шифрования данных DEA (Data Encryption Algorithm), а ISO - DEA-1, за 20 лет стал мировым стандартом. Хотя на нем и появился налет старости, он весьма прилично выдержал годы криптоанализа и все еще остается безопасным по отношению ко всем врагам, кроме, возможно, самых могущественных.

Описание DES

DES представляет собой блочный шифр, он шифрует данные 64-битовыми блоками. С одного конца алгоритма вводится 64-битовый блок открытого текста, а с другого конца выходит 64-битовый блок шифротекста. DES является симметричным алгоритмом: для шифрования и дешифрирования используются одинаковые алгоритм и ключ (за исключением небольших различий в использовании ключа).

Длина ключа равна 56 битам. (Ключ обычно представляется 64-битовым числом, но каждый восьмой бит используется для проверки четности и игнорируется. Биты четности являются наименьшими значащими битами байтов ключа.) Ключ, который может быть любым 56-битовым числом, можно изменить в любой момент времени. Ряд чисел считаются слабыми ключами, но их можно легко избежать. Безопасность полностью определяется ключом.

На простейшем уровне алгоритм не представляет ничего большего, чем комбинация двух основных методов шифрования: смещения и диффузии. Фундаментальным строительным блоком DES является применение к тексту единичной комбинации этих методов (подстановка, а за ней - перестановка), зависящей от ключа. Такой блок называется этапом. DES состоит из 16 этапов, одинаковая комбинация методов применяется к открытому тексту 16 раз (см. Рис. 1.).

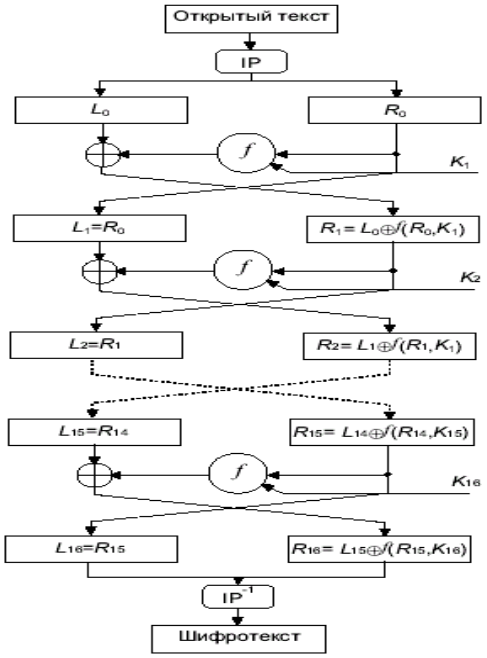


Рис. 1. DES.

Алгоритм использует только стандартную арифметику 64-битовых чисел и логические операции, поэтому он легко реализовывался в аппаратуре второй половины *10-х.* Изобилие повторений в алгоритме делает его идеальным для реализации в специализированной микросхеме. Первоначальные программные реализации были

довольно неуклюжи, но сегодняшние программы намного лучше.

Схема алгоритма

DES работает с 64-битовым блоком открытого текста. После первоначальной перестановки блок разбивается на правую и левую половины длиной по 32 бита. Затем выполняется 16 этапов одинаковых действий, называемых функцией f, в которых данные объединяются с ключом. После шестнадцатого этапа правая и левая половины объединяются и алгоритм завершается заключительной перестановкой (обратной по отношению к первойачальной).

На каждом этапе (см. 10-й) биты ключа сдвигаются, и затем из 56 битов ключа выбираются 48 битов. Правая половина данных увеличивается до 48 битов с помощью перестановки с расширением, объединяется посредством XOR с 48 битами смещенного и переставленного ключа, проходит через 8 S-блоков, образуя 32 новых бита, и переставляется снова. Эти четыре операции и выполняются функцией f Затем результат функции f объединяется с левой половиной с помощью другого XOR. В итоге этих действий появляется новая правая половина, а старая правая половина становится новой левой. Эти действия повторяются 16 раз, образуя 16 этапов DES.

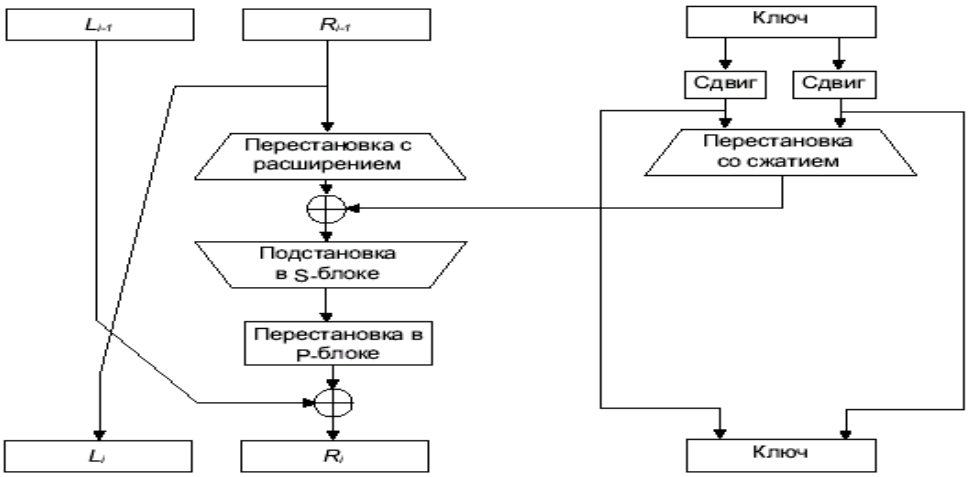


Рис. 2. Один этап DES.

Если *BJ* - это результат г-ой итерации, L_t и R_t - левая и правая половины B_t, K_t - 48-битовый ключ для этапа *i,* а f - это функция, выполняющие все подстановки, перестановки и XOR с ключом, то этап можно представить как начальная перестановка

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i)$$

Начальная перестановка выполняется еще до этапа 1, при этом входной блок переставляется, как показано в Табл. 1. Эту и все другие таблицы надо читать слева направо и сверху вниз. Например, начальная перестановка перемещает бит 58 в битовую позицию 1, бит 50 - в битовую позицию 2, бит 42 - в битовую позицию 3, и так далее.

Табл. 1. Начальная перестановка

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,	И,	3,
61,	53,	45,	37,	29,	21,	13,	5,	63,	55,	47,	39,	31,	23,	15,	7

Начальная перестановка и соответствующая заключительная перестановка не влияют на безопасность DES. (Как можно легко заметить, эта перестановка в первую очередь служит для облегчения побайтной загрузки данных открытого текста и шифротекста в микросхему DES. Не забывайте, что DES появился раньше 16- и 32-битовых микропроцессорных шин.) Так как программная реализация этой многобитовой

перестановки нелегка (в отличие от тривиальной аппаратной), во многих программных реализациях DES начальная и заключительные перестановки не используются. Хотя такой новый алгоритм не менее безопасен, чем DES, он не соответствует стандарту DES и, поэтому, не может называться DES.

Преобразования ключа

Сначала 64-битовый ключ DES уменьшается до 56-битового ключа отбрасыванием каждого восьмого бита. Эти биты используются только для контроля четности, позволяя проверять правильность ключа. После извлечения 56-битового ключа для каждого из 16 этапов DES генерируется новый 48-битовыйподключ**.** Эти подключи, *Ki,* определяются следующим образом.

Табл. 2. Перестановка ключа

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

Во первых, 56-битовый ключ делится на две 28-битовых половинки. Затем, половинки циклически сдвиг а-ются налево на один или два бита в зависимости от этапа. Этот сдвиг показан в 9-й.

Табл. 3. Число битов сдвига ключа в зависимости от этапа

Этап	1	1	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Число	2	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

После сдвига выбирается 48 из 56 битов. Так как при этом не только выбирается подмножество битов, но и изменяется их порядок, эта операция называется перестановка со сжатием**.** Ее результатом является набор из 48 битов. Перестановка со сжатием (также называемая переставленным выбором) определена в 8-й. Например, бит сдвинутого ключа в позиции 33 перемещается в позицию 35 результата, а 18-й бит сдвинутого ключа отбрасывается.

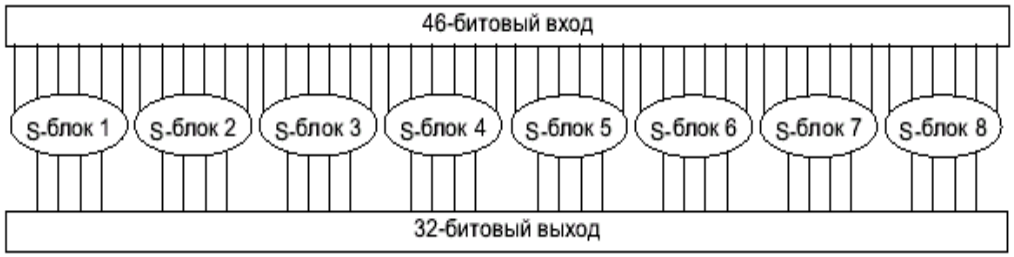
Табл. 4. Перестановка со сжатием

14,	17,	11,	24,	1,	5,	3,	28,	15,	6,	21,	Ю,
23,	19,	11**,**	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32

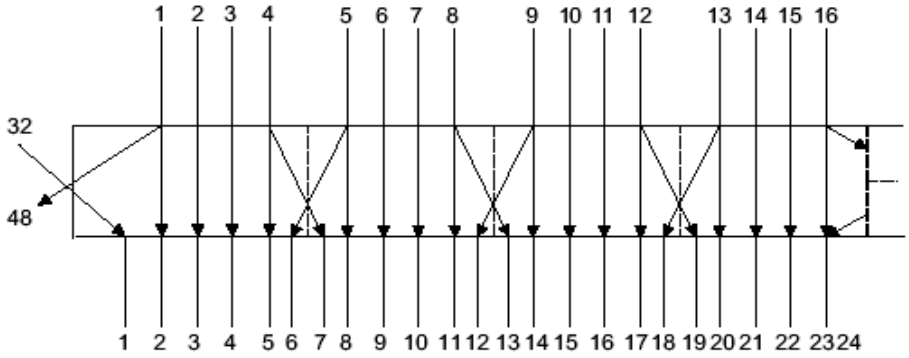
Из-за сдвига для каждого подключа используется отличное подмножество битов ключа. Каждый бит используется приблизительно в 14 из 16 подключей, хотя не все биты используются в точности одинаковое число раз.

Перестановка с расширением

Эта операция расширяет правую половину данных, R_t, от 32 до 48 битов. Так как при этом не просто повторяются определенные биты, но и изменяется их порядок, эта операция называется перестановкой с расширением. У нее две задачи: привести размер правой половины в соответствие с ключом для операции XOR и получить более длинный результат, который можно будет сжать в ходе операции подстановки. Однако главный криптографический смысл совсем в другом. За счет влияния одного бита на две подстановки быстрее возрастает зависимость битов результата от битов исходных данных. Это называется лавинным эффектом. DES спроектирован так, чтобы как можно быстрее добиться зависимости каждого бита шифротекста от каждого бита открытого текста и каждого бита ключа.



Перестановка с расширением показана на Рис. 3. Иногда она называется E-блоком (от expansion). Для каждого 4-битового входного блока первый и четвертый бит представляют собой два бита выходного блока, а второй и третий биты - один бит выходного блока. В 7-й



показано, какие позиции результата соответствуют каким позициям исходных данных. Например, бит входного блока в позиции 3 переместится в позицию 4 выходного блока, а бит входного блока в позиции 21 - в позиции 30 и 32 выходного блока.

Рис. 3. Перестановка с расширением.

Хотя выходной блок больше входного, каждый входной блок генерирует уникальный выходной блок.

Табл. 5. Перестановка с расширением

3	1,	2,	3,	4,	5,	4,	5,	6,	7,	8,	9,
2,											
8,	9,	1	1	1	1	1	1	1	1	1	1
		0,	1,	2.,	3,	2,	3,	4,	5,	6,	7,
1	1	1	1	2	2	2	2	2	2	2	2
6,	7,	8,	9,	0,	1,	0,	1,	2,	3,	4,	5,
2	2	2	2	2	2	2	2	3	3	3	1
4,	5,	6,	7,	8,	9,	8,	9,	0,	1,	2,	

Подстановка с помощью S-блоков

После объединения сжатого блока с расширенным блоком с помощью XOR над 48-битовым результатом выполняется операция подстановки. Подстановки производятся в восьми блоках подстановки**,** или S-блоках (от substitution). У каждого S-блока 6-битовый вход и 4-битовый выход, всего используется восемь различных S-блоков. (Для восьми S-блоков DES потребуется 256 байтов памяти.) 48 битов делятся на восемь 6-битовых подблока. Каждый отдельный подблок обрабатывается отдельным S-блоком: первый подблок - S-блоком 1, вт о-рой - S-блоком 2, и так далее. См. Рис. 4.

Рис. 4. Подстановка - S-блоки.

Каждый S-блок представляет собой таблицу из 2 строк и 16 столбцов. Каждый элемент в блоке является 4-битовым числом. По 6 входным битам S-блока определяется, под какими номерами столбцов и строк искать выходное значение. Все восемь S-блоков показаны в Табл. 6.

Табл. 6. S-блоки

4,	1,	14,	8,	13,	6,	2,	11,	15,	12,	9,	7,	3,	10,	5,	0,
15,	12,	8,	2,	4,	9,	1,	7,	5,	11,	3,	14,	10,	0,	6,	13,
S-блок 2:															
15,	1,	8,	14,	6,	11,	3,	4,	9,	7,	2,	13,	12,	0,	5,	10,
3,	13,	4,	7,	15,	2,	8,	14,	12,	0,	1,	10,	6,	9,	11,	5,
0,	14,	7,	11,	10,	4,	13,	1,	5,	8,	12,	6,	9,	3,	2,	15,
13,	8,	10,	1,	3,	15,	4,	2,	11,	6,	7,	12,	0,	5,	14,	9,
S-блок 3:															
10,	0,	9,	14,	6,	3,	15,	5,	1,	13,	12,	7,	11,	4,	2,	8,
13,	7,	0,	9,	3,	4,	6,	10,	2,	8,	5,	14,	12,	11,	15,	1,
13,	6,	4,	9,	8,	15,	3,	0,	11,	1,	2,	12,	5,	10,	14,	7,
1,	10,	13,	0,	6,	9,	8,	7,	4,	15,	14,	3,	11,	5,	2,	12,
S-блок 4:															
7,	13,	14,	3,	0,	6,	9,	10,	1,	2,	8,	5,	11,	12,	4,	15,
13,	8,	11,	5,	6.	15,	0,	3,	4,	7,	2,	12,	1,	10,	14,	9,
10,	6,	9,	0,	12,	11,	7,	13,	15,	1,	3,	14,	5,	2,	8,	4,
3,	15,	0,	6,	10,	1,	13,	8,	9,	4,	5,	11,	12,	7,	2,	14,
S-блок 5:															
2,	12,	4,	1,	7,	10,	11,	6,	8,	5,	3,	15,	13,	0,	14,	9,
14,	11,	2,	12,	4,	7,	13,	1,	5,	0,	15,	10,	3,	9,	8,	6,
4,	2,	1,	11,	10,	13,	7,	8,	15,	9,	12,	5,	6,	3,	0,	14,
11,	8,	12,	7,	1,	14,	2,	13,	6,	15,	0,	9,	10,	4,	5,	3,
S-блок 6:															
12,	1,	10,	15,	9,	2,	6,	8,	0,	13,	3,	4,	14,	7,	5,	11,
10,	15,	4,	2,	7,	12,	9,	5,	6,	1,	13,	14,	0,	11,	3,	8,
9,	14,	15,	5,	2,	8,	12,	3,	7,	0,	4,	10,	1,	13,	11,	6,
4,	3,	2,	12,	9,	5,	15,	10,	11,	14,	1,	7,	6,	0,	8,	13,
S-блок 7:															
4,	11,	2,	14,	15,	0,	8,	13,	3,	12,	9,	7,	5,	10,	6,	1,
13,	0,	11,	7,	4,	9,	1,	10,	14,	3,	5,	12,	2,	15,	8,	6,
1,	4,	11,	13,	12,	3,	7,	14,	10,	15,	6,	8,	0,	5,	9,	2,
6,	11,	13,	8,	1,	4,	10,	7,	9,	5,	0,	15,	14,	2,	3,	12,
S-блок 8:															
13,	2,	8,	4,	6,	15,	11,	1,	10,	9,	3,	14,	5,	0,	12,	7,
1,	15,	13,	8,	10,	3,	7,	4,	12,	5,	6,	11,	0,	14,	9,	2,
7,	11,	4,	1,	9,	12,	14,	2,	0,	6,	10,	13,	15,	3,	5,	8,
2,	1,	14,	7,	4,	10,	8,	13,	15,	12,	9,	0,	3,	5,	6,	11

Входные биты особым образом определяют элемент S-блока. Рассмотрим 6-битовый вход S-блока: *Ь\, Ь₂, Ьз, bo,, bs* и *bf,*. Биты *b* и *bf,* объединяются, образуя 2-битовое число от 0 до 3, соответствующее строке таблицы. Средние 4 бита, с *Ь₂* по *Ь₅,* объединяются, образуя 4-битовое число от 0 до 15, соответствующее столбцу таблицы.

Например, пусть на вход шестого S-блока (т.е., биты функции XOR с 31 по 36) попадает 110011. Первый и последний бит, объединяясь, образуют 11, что соответствует строке 3 шестого S-блока. Средние 4 бита образуют 1001, что соответствует столбцу 9 того же S-блока. Элемент S-блока 6, находящийся на пересечении строки 3 и столбца 9, - это 14. (Не забывайте, что строки и столбцы нумеруются с 0, а не с 1.) Вместо 110011 подставляется 1110.

Конечно же, намного легче реализовать S-блоки программно в виде массивов с 64 элементами. Для этого потребуется переупорядочить элементы, что не является трудной задачей. (Изменить индексы, не изменяя порядок элементов, недостаточно. S-блоки спроектированы очень тщательно.) Однако такой способ описания S-блоков помогает понять, как они работают. Каждый S-блок можно рассматривать как функцию подстановки 4-битового элемента: *Ъг* появляются входом, а некоторое 4-битовое число –

результатом. Биты Ъ\ и $Ь_6$ определяются соседними блоками, они определяют одну из четырех функций подстановки, возможных в данном S-блоке.

Подстановка с помощью S-блоков является ключевым этапом DES. Другие действия алгоритма линейны и легко поддаются анализу. S-блоки нелинейны, и именно они в большей степени, чем все остальное, обеспечивают безопасность DES.

В результате этого этапа подстановки получаются восемь 4-битовых блоков, которые вновь объединяются в единый 32-битовый блок. Этот блок поступает на вход следующего этапа - перестановки с помощью P-блоков.

Перестановка с помощью P-блоков

32-битовый выход подстановки с помощью S-блоков, перетасовываются в соответствии с P-блоком. Эта перестановка перемещает каждый входной бит в другую позицию, ни один бит не используется дважды, и ни один бит не игнорируется. Этот процесс называется прямой перестановкой или просто перестановкой. Позиции, в которые перемещаются биты, показаны в 5-й. Например, бит 21 перемещается в позицию 4, а бит 4 - в позицию 31.

Табл. 7. Перестановка с помощью P-блоков

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

Наконец, результат перестановки с помощью P-блока объединяется посредством XOR с левой половиной первоначального 64-битового блока. Затем левая и правая половины меняются местами, и начинается следующий этап.

Заключительная перестановка

Заключительная перестановка является обратной по отношению к начальной перестановки. Обратите внимание, что левая и правая половины не меняются местами после последнего этапа DES, вместо этого объединенный блок $Л_{16}£_{16}$ используется как вход заключительной перестановки. В этом нет ничего особенного, перестановка половинок с последующим циклическим сдвигом

привела бы к точно такому же результату. Это сделано для того, чтобы алгоритм можно было использовать как для шифрования, так и для дешифрирования.

Табл. 8. Заключительная перестановка

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

Дешифрирование DES

После всех подстановок, перестановок, операций XOR и циклических сдвигов можно подумать, что алгоритм дешифрирования, резко отличаясь от алгоритма шифрования, точно также запутан. Напротив, различные компоненты DES были подобраны так, чтобы выполнялось очень полезное свойство: для шифрования и дешифрирования используется один и тот же алгоритм.

DES позволяет использовать для шифрования или дешифрирования блока одну и ту же функцию. Единственное отличие состоит в том, что ключи должны использоваться в обратном порядке. То есть, если на этапах шифрования использовались ключи $K1, K_2, K_3, ..., K_{16}$, то ключами дешифрирования будут $K_{16}, K_{15}, K_{i4}, ..., K1$. Алгоритм, который создает ключ для каждого этапа, также цикличен. Ключ сдвигается направо, а число поз и-ций сдвига равно 0, 1,2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

Режимы DES

PIPS PUB 81 определяет четыре режима работы: ECB, CBC, OFB и CPB. Банковские стандарты ANSI определяют для шифрования ECB и CBC, а для проверки подлинности - CBC и n-битовый CPB .

В мире программного обеспечения сертификация обычно не важна. Из-за своей простоты в большинстве существующих коммерческих программ используется ECB, хотя этот режим наиболее чувствителен к вскрытию. CBC используется редко несмотря на то, что он лишь незначительно сложнее, чем ECB, и обеспечивает большую безопасность.

Безопасность DES

Специалисты давно интересуются безопасностью DES. Было много рассуждений о длине ключа, количестве итераций и схеме S-блоков. S-блоки были наиболее таинственными - какие-то константы, без видимого объяснения для чего и зачем они нужны. Хотя IBM утверждала, что работа алгоритма была результатом 17 человеко-лет интенсивного криптоанализа, были высказаны опасения, что NSA вставило в алгоритм лазейку, которая позволит агентству легко дешифрировать перехваченные сообщения.

Комитет по разведке Сената США чрезвычайно тщательно расследовал этот вопрос в 1978 году. Результаты работы комитета были засекречены, но в открытых итогах этого расследования с NSA были сняты все обвинения в неуместном вмешательстве в проектирование алгоритма. "Было сказано, что NSA

убедило IBM в достаточности более короткого ключа, косвенно помогло разработать структуры S-блоков и подтвердило, что в окончательном варианте DES, с учетом всех знаний NSA, отсутствовали статистические или математические бреши ". Однако, так как правительство не опубликовало подробности расследования, убедить многих не удалось.

Слабые ключи

Из-за того, что первоначальный ключ изменяется при получении подключа для каждого этапа алгоритма, определенные первоначальные ключи являются слабыми. Вспомните, первоначальное значение расщепляется на две половины, каждая из которых сдвигается независимо. Если все биты каждой половины равны 0 или 1, то для всех этапов алгоритма используется один и тот же ключ. Это может произойти, если ключ состоит из одних 1, из одних 0, или если одна половина ключа состоит из одних 1, а другая - из одних 0. Кроме того, у два слабых ключа обладают другими свойствами, снижающими их безопасность.

Четыре слабых ключа показаны в шестнадцатиричном виде в Табл. 11. (Не забывайте, что каждый восьмой бит -это бит четности.).

Табл. 11. Слабые ключи DES

Значение слабого ключа (с битами четн ости)			Действительный ключ
0101	0101	0101	010 1 0000000 0000000
1F1F	1F1F	OEOE	OE OE 0000000 FFFFFFF
EOE O	EOEO	F1F1	F1F 1 FFFFFFF 0000000
FEFE	FEFE	FEFE	FEF E FFFFFFF FFFFFFF

Кроме того, некоторые пары ключей при шифровании переводят открытый текст в идентичный шифротекст. Иными словами, один из ключей пары может расшифровать сообщения, зашифрованные другим ключом пары. Это происходит из-за метода, используемого DES для генерации подключей - вместо 16 различных подключей эти ключи генерируют только два различных подключа. В алгоритме каждый из этих подключей используется восемь раз. Эти ключи, называемые полуслабыми ключами**,** в шестнадцатиричном виде приведены в Табл. 12.

Табл. 12. Полуслабые пары ключей DES

01 FE	01 FE	01 FE	01 FE	и 1	FE0 01	FE 01	FE 01	FE
1F EO	1F EO	OE F1	OE F1	и F	E01 1F	E0 0E	F1 E	F10
01 EO	01 EO	01 F1	01 F1	и 1	E00 01	E0 01	F1 1	F10
1F IF	IE 1F	OE 0E	OE 0E	и	FE1 F 1F01 01	FE OE 1F OE 01	FE OE OE 01	FE

E EO FI FIF и FEE FE FE FE
OFE FE FE E O EO E1 E1

Ряд ключей генерирует только четыре подключа, каждый из которых четыре раза используется в алгоритме.Эти возможно слабые ключи перечислены в Табл. 13.

Табл. 13. Возможно слабые ключи DES

I	I	0	0	C	C	0	0	E	0	0	E	F	0	0	F
F	F	1	1	E	E	1	1	O	1	1	O	I	1	1	I
0	I	I	0	0	C	C	0	F	I	0	E	F	C	0	F
1	F	F	1	1	E	E	1	E	F	1	O	E	E	1	I
I	0	0	I	C	0	0	C	F	0	I	E	F	0	C	F
F	1	1	F	E	1	1	E	E	1	F	O	E	1	E	I
0	0	I	I	0	0	C	C	E	I	I	E	F	C	C	F
1	1	F	F	1	1	E	E	O	F	F	O	I	E	E	I
E	E	0	0	F	F	0	0	F	0	0	F	F	0	0	F
O	O	1	1	I	I	1	1	E	1	1	E	E	1	1	E
F	F	0	0	F	F	0	0	E	I	0	F	F	C	0	F
E	E	1	1	E	E	1	1	O	F	1	E	I	E	1	E
F	E	I	0	F	F	C	0	E	0	I	F	F	0	C	F
E	O	F	1	E	I	E	1	O	1	F	E	I	1	E	E
E	F	I	0	F	F	C	0	F	I	I	F	F	C	C	F
O	E	F	1	I	E	E	1	E	F	F	E	E	E	E	E
F	E	0	I	F	F	0	C	I	F	0	E	C	F	0	F
E	O	1	F	E	I	1	E	F	E	1	O	E	E	1	I
E	F	0	I	F	F	0	C	0	F	I	E	0	F	C	F
O	E	1	F	I	E	1	E	1	E	F	O	1	E	E	I
E	E	I	I	F	F	C	C	I	E	0	F	C	F	0	F
O	O	F	F	I	I	E	E	F	O	1	E	E	I	1	E
F	F	I	I	F	F	C	C	0	E	I	F	0	F	C	F
E	E	F	F	E	E	E	E	1	O	F	E	1	I	E	E
F	I	E	0	F	C	F	0	0	0	E	E	0	0	F	F
E	F	O	1	E	E	I	1	1	1	O	O	1	1	I	I
E	I	F	0	F	C	F	0	I	I	E	E	C	C	F	F
O	F	E	1	I	E	E	1	F	F	O	O	E	E	I	I
F	0	E	I	F	0	F	C	I	0	F	E	C	0	F	F
E	1	O	F	E	1	I	E	F	1	E	O	E	1	E	I
E	0	F	I	F	0	F	C	0	I	F	E	0	C	F	F
O	1	E	F	I	1	E	E	1	F	E	O	1	E	E	I
0	E	E	0	0	F	F	0	I	0	E	F	C	0	F	F
1	O	O	1	1	I	I	1	F	1	O	E	E	1	I	E
I	F	E	0	C	F	F	0	0	I	E	F	0	C	F	F
F	E	O	1	E	E	O	1	1	F	O	E	1	E	I	E
I	E	F	0	C	F	F	0	0	0	F	F	0	0	F	F
F	O	E	1	E	I	E	1	1	1	E	E	1	1	E	E
0	F	F	0	0	F	F	0	I	I	F	F	C	C	F	F
1	E	E	1	1	E	E	1	F	F	E	E	E	E	E	E
I	E	E	I	C	F	F	C	F	F	E	E	F	F	F	F
F	O	O	F	E	I	I	E	E	E	O	O	E	E	I	I

0 F E I 0 F F C E F F E F F F F
1 E O F 1 E I E O E E O I E E I
0 E F I 0 F F C F E E F F F F F
1 O E F 1 I E E E O O E E I I E
I F F I C F F C E E F F F F F F
F E E F E E E E O O E E I I E E

Прежде, чем порицать DES слабые ключи, обратите внимание на то, что эти 64 ключа - это крошечная часть полного набора из 72057594037927936 возможных ключей. Если вы выбираете ключ случайно, вероятность выбрать один из слабых ключей пренебрежимо мала. Вы можете всегда проверять "на слабость" сгенерированный ключ. Некоторые думают, что нечего и беспокоиться на этот счет. Другие утверждают, что проверка очень легка, почему бы ее и не выполнить. Других слабых ключей в процессе и с-следований найдено не было.

Количество этапов

Почему 16 этапов? Почему не 32? После пяти этапов каждый бит шифротекста является функцией всех б и-тов открытого текста и всех битов ключа [1078, 1080], а после восьми этапов шифротекст по сути представляет собой случайную функцию всех битов открытого текста и всех битов ключа. (Это называется лавинным эффектом.) Так почему не остановиться после восьми этапов?

В течение многих лет версии DES с уменьшенным числом этапов успешно вскрывались. DES с тремя и четырьмя этапами был легко взломан в 1982 году. DES с шестью этапами пал несколькими годами позже. Дифференциальный криптоанализ Бихама и Шамира объяснил и это: DES с любым количеством этапов, меньшим 16, может быть взломан с помощью вскрытия с известным открытым текстом быстрее, чем с помощью вскрытия грубой силой. Конечно грубый взлом является более вероятным способом вскрытия, но интересен тот факт, что алгоритм содержит ровно 16 этапов.

Задание.

1. Ознакомиться с алгоритмом DES.
2. Установить программу DES.
3. Произвести шифрование контрольной фразы.
4. Оценить время шифрования и затраченные ресурсы компьтера.
5. Произвети расшифрование контрольной фразы.
6. Оценить затраченные ресурсы и время.

А л г о р и т м R S A

Безопасность RSA основана на трудности разложения на множители больших чисел . Открытый и закрытый ключи являются функциями двух больших (100 - 200 разрядов или даже больше) простых чисел. Предполагается, что восстановление открытого текста по шифротексту и открытому ключу эквивалентно разложению на множители двух больших чисел .

Для генерации двух ключей используются два больших случайных простых числа , *p* и *q.* Для максимальной безопасности выбирайте *p* и *q* равной длины. Рассчитывается произведение:

$n = pq$

Затем случайным образом выбирается ключ шифрования *e,* такой что *e* и (p-1)(q-1) являются взаимно простыми числами. Наконец расширенный алгоритм Эвклида используется для вычисления ключа дешифрирования *d,* такого что

$ed = 1 (mod (p-1)(q-1))$ Другими словами

$d = e^{-1} mod ((p-1)(q-1))$

Заметим, что *d* и *n* также взаимно простые числа. Числа *e* и *n* - это открытый ключ, а число *d* - закрытый. Два простых числа *p* и *q* больше не нужны. Они должны быть отброшены, но не должны быть раскрыты .

Для шифрования сообщения *m* оно сначала разбивается на цифровые блоки, меньшие *n* (для двоичных данных выбирается самая большая степень числа 2, меньшая *n).* То есть, если *p* и *q* - 100-разрядные простые числа, то *n* будет содержать около 200 разрядов, и каждый блок сообщения то *n*, должен быть около 200 разрядов в длину. (Если нужно зашифровать фиксированное число блоков, их можно дополнить несколькими нулями слева, чтобы гарантировать, что блоки всегда будут меньше *n.* Зашифрованное сообщение *c* будет состоять из блоков c_t той же самой длины . Формула шифрования выглядит так

$c_t = m_i^{\,e} \bmod n.$

Для расшифровки сообщения возьмите каждый зашифрованный блок c_i, и вычислите

$m_i = c_t^{\,d} \bmod n$

Так как

$C^d = (m_i^{\,e})^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i$; все (mod *n*)

формула восстанавливает сообщение.

Табл. 1.

Ш и ф р о в а н и е R S A

Открытый ключ:

n произведение двух простых чисел *p* и *q (p* и *q* должны храниться в секрете)

e число, взаимно простое с *(p*-1*)(q*-1*)*

Закрытый ключ:

d $e^{-1} \bmod (p-1)(q-1)$

Шифрование:

$c = m^e \bmod n$

Дешифрирование:

$m = c^d \bmod n$

Точно также сообщение может быть зашифровано с помощью *d,* а расшифровано с помощью *e,* возможен любой выбор.

Короткий пример поможет пояснить работу алгоритма . Если $p = 47$ q = *11,* то

$n=pq=3337$

Ключ *e* не должен иметь общих множителей

$(p-1)(q-1)= 46*70 =3220$

Выберем (случайно) *e* равным 79. В этом случае $d= 79^{-1} \bmod 3220 = 1019$

При вычислении этого числа использован расширенный алгоритм Эвклида. Опубликуем *e* и *n,* сохранив в секрете *d.* Отбросим p и *q.* Для шифрования сообщения

m = 6882326879666683

сначала разделим его на маленькие блоки . Для нашего случая подойдут трехбуквенные блоки. Сообщение разбивается на шесть блоков m_i.

$m_1 = 688$

$m_2 = 232$

$m_3 = 687$

$m_4 = 966$

$m_5 = 668$

$m_6 = 003$

Первый блок шифруется как $688^{79} \bmod 3337 = 1570 = c_1$

Выполняя те же операции для последующих блоков, создает шифротекст сообщения :

c = 1570 2756 2091 2276 2423 158

Для дешифрирование нужно выполнить такое же возведение в степень, используя ключ дешифрирования 1019:

$1570^{1019} \bmod 3337 = 688 = m_1$

Аналогично восстанавливается оставшаяся часть сообщения .

Скорость RSA

Аппаратно RSA примерно в 1000 раз медленнее DES. Скорость работы самой быстрой СБИС-реализации RSA с 512-битовым модулем - 64 килобита в секунду . Существуют также микросхемы, которые выполняют 1024-битовое шифрование RSA. В настоящее время разрабатываются микросхемы, которые, используя 512-битовый модуль, приблизятся к рубежу 1 Мбит/с. Производители также применяют RSA в интеллектуальных карточках, но эти реализации медленнее.

Программно DES примерно в 100 раз быстрее RSA. Эти числа могут незначительно измениться при изменении технологии, но RSA никогда не достигнет скорости симметричных алгоритмов.

Табл. 2.

Скорости RSA для различных длин модулей при 8-битовом открытом ключе (на SPARC II)

	512	768	1024 бита

Шифрование	0.03с		0.05с		0.08с	0.93с
Дешифрирование	0.16с	0.16с	0.48с	0.52с	0.97с	0.08с
Подпись Проверка	0.02с		0.07с			

Программные Speedups

Шифрование RSA выполняется намного быстрей, если правильно выбрать значение *e.* Тремя наиболее частыми вариантами являются 3, 17 и 65537 (2^{16} + 1). (Двоичное представление 65537 содержит только две единицы, поэтому для возведения в степень нужно выполнить только 17 умножений.) X.509 советует 65537, PEM рекомендует 3, а PKCS # - 3 или 65537. Не существует никаких проблем безопасности, связанных с использованием в качестве *e* любого из этих трех значений (при условии, что вы дополняете сообщения случайными числами), даже если одно и то же значение *e* используется целой группой пользователей.

Операции с закрытым ключом можно ускорить при помощи китайской теоремы от остатках, если вы сохр а-нили значения *p* и *q,* а также дополнительные значения: *d* mod *(p* - 1), *d* mod *(q* - 1) и ∂^{n} *modp* [1283, 1276]. Эти дополнительные числа можно легко вычислить по закрытому и открытому ключам .

Безопасность RSA

Безопасность RSA полностью зависит от проблемы разложения на множители больших чисел . Технически, это утверждение о безопасности некорректно. Предполагается, что безопасность RSA зависит от проблемы разложения на множители больших чисел. Никогда не было доказано математически, что нужно разложить *n* на множители, чтобы восстановить *m* по *c* и *e.* Понятно, что может быть открыт совсем иной способ криптоанализа RSA. Однако, если этот новый способ позволит криптоаналитику получить *d,* он также может быть использован для разложения на множители больших чисел.

Также можно вскрыть RSA, угадав значение *(p*-1*)(q*-1*).* Это вскрытие не проще разложения *n* на множители .

Для сверхскептиков: доказано, что некоторые варианты RSA также сложны, как и разложение на множители . Раскрытие даже нескольких битов информации по зашифрованному RSA шифротексту не легче, чем дешифрирование всего сообщения .

Самым очевидным средством вскрытия является разложение *n* на множители. Любой противник сможет получить открытый ключ *e* и модуль *n.* Чтобы найти ключ дешифрирования *d,* противник должен разложить *n* на множители. В настоящее время передним краем этой технологии является число, содержащее 129 десятичных цифр . Значит, *n* должно быть больше этого значения.

Конечно, криптоаналитик может перебирать все возможные *d,* пока он не подберет правильное значение. Но такое вскрытие грубой силой даже менее эффективно, чем попытка разложить *n* на множители.

Вскрытие с выбранным шифротекстом против RSA

Некоторые вскрытия работают против реализаций RSA. Они вскрывают не сам базовый алгоритм, а надстроенный над ним протокол. Важно понимать, что само по себе использование RSA не обеспечивает безопасности. Дело в реализации.

Сценарий 1: Еве, подслушавшей линии связи Алисы, удалось перехватить сообщение *c,* шифрованное с помощью RSA открытым ключом Алисы. Ева хочет прочитать сообщение. На языке математики, ей нужно *m,* для которого

$m = c^d$

Для раскрытия *m* она сначала выбирает первое случайное число r, меньшее *n.* Она достает открытый ключ Алисы *e.* Затем она вычисляет

$x = f \bmod n$

$y = xc \bmod n$

$t = r^{-1} \bmod n$

Если $x = r^e \bmod n$, то $r = x^d \bmod n$.

Теперь просит Алису подписать *y* ее закрытым ключом, таким образом расшифровав *y.* (Алиса должна подписать сообщение.) Не забывайте, Алиса никогда раньше не видела *y.* Алиса посылает Еве

$u = y^d \bmod n$

Теперь Ева вычисляет

$tu \bmod n = r^{-1} y^d \bmod n = r^{-1} x^d c^d \bmod n = c^d \bmod n = m$

И Ева получает *m.*

Никогда не пользуйтесь алгоритмом RSA для подписи случайных документов, подсунутых вам посторонними.

Выводы

— Знание одной пары показателей шифрования/дешифрирования для данного модуля позволяет взломщику разложить модуль на множители.

— Знание одной пары показателей шифрования/дешифрирования для данного модуля позволяет взломщику вычислить другие пары показателей, не раскладывая модуль на множители.

— В протоколах сетей связи, применяющих RSA, не должен использоваться общий модуль. (Это является быть очевидным следствием предыдущих двух пунктов.)

— Для предотвращения вскрытия малого показателя шифрования сообщения должны быть дополнены случайными значениями.

— Показатель дешифрирования должен быть большим.

Не забывайте, недостаточно использовать безопасный криптографический алгоритм, должны быть безопасными вся криптосистема и криптографический протокол. Слабое место любого из трех этих компонентов сделает небезопасной всю систему.

З а д а н и е .

7. Ознакомиться с алгоритмом RSA.
8. Разработать программу RSA.
9. Произвести шифрование контрольной фразы.
10. Оценить время шифрования и затраченные ресурсы компьтера.
11. Произвети расшифрование контрольной фразы.
12. Оценить затраченные ресурсы и время.

АЛГОРИТМЫ ИДЕНТИФИКАЦИИ

В своих работах [544, 545], Фейге, Фиат и Шамир показали, как параллельная схема может повысить число аккредитаций на этап и уменьшить взаимодействия Пегги и Виктора .

Сначала, как и в предыдущем примере, генерируется n, произведение двух больших простых чисел. Для генерации открытого и закрытого ключей Пегги сначала выбирается k различных чисел: $v_1, v_2, \ldots v_k$, где каждое v_i является квадратичным остатком mod n. Иными словами, v_i выбираются так, чтобы $x^2 \equiv v_i \pmod n$ имело решение, и существовало $v_i^{-1} \bmod n$. Строка, $v_1, v_2, \ldots v_k$, служит открытым ключом. Затем вычисляются наименьшие s_i, для которых $s_i \equiv \text{sqrt}\ (v_i^{-1}) \pmod n$. Строка $s_1, s_2, \ldots s_k$, служит закрытым ключом.

Выполняется следующий протокол:

(1) Пегги выбирает случайное r, меньшее n. Затем она вычисляет $x = -r^2 \bmod n$ и посылает x Виктору.

(2) Виктор посылает Пегги строку из k случайных битов: $b_1, b_2, \ldots b_k$.

(3) Пегги вычисляет $y = r * (s_1^{b_1} * s_2^{b_2} * \ldots * s_k^{b_k}) \bmod n$. (Она перемножает вместе значения s_i, соответствующие b_i=1. Если первым битом Виктора будет 1, то s_1 войдет в произведение, а если первым битом будет 0, то нет, и т.д.) Она посылает y Виктору.

(4) Виктор проверяет, что $x = y^2 * (v_1^{b_1} * v_2^{b_2} * \ldots * v_k^{b_k}) \bmod n$. (Он перемножает вместе значения v_i, основываясь га случайной двоичной строке. Если его первым битом является 1, то v_1 войдет в произведение, а если первым битом будет 0, то нет, и т.д.)

Пегги и Виктор повторяют этот протокол t раз, пока Виктор не убедится, что Пегги знает $s_1, s_2, \ldots s_k$.

Вероятность, что Пегги удастся обмануть Виктор t раз, равна $1/2^{kt}$. Авторы рекомендуют использовать вероятность мошенничества $1/2^{20}$ и предлагают значения $k = 5$ и $t = 4$. Если у вас склонность к мании преследования, увеличьте эти значения.

Пример

Взглянем на работу этого протокола небольших числах . Если $n = 35$ (два простых числа - 5 и 7), то возможными квадратичными остатками являются:

1: $x^2 \equiv 1 \pmod{35}$ имеет решения: $x = 1, 6, 29, 34$.

4: $x^2 \equiv 4 \pmod{35}$ имеет решения: $x = 2, 12, 23, 33$.

9: $x^2 \equiv 9 \pmod{35}$ имеет решения: $x = 3, 17, 18, 32$.

11: $x^2 \equiv 11 \pmod{35}$ имеет решения: $x = 9, 16, 19, 26$.

14: $x^2 \equiv 14 \pmod{35}$ имеет решения: $x = 7, 28$.

15: $x^2 \equiv 15 \pmod{35}$ имеет решения: $x = 15, 20$.

16: $x^2 \equiv 16 \pmod{35}$ имеет решения: $x = 4, 11, 24, 31$.

21: $x^2 \equiv 21 \pmod{35}$ имеет решения: $x = 14, 21$.

25: $x^2 \equiv 25 \pmod{35}$ имеет решения: $x = 5, 30$.

29: $x^2 \equiv 29 \pmod{35}$ имеет решения: $x = 8, 13, 22, 27$.

30: $x^2 \equiv 30 \pmod{35}$ имеет решения: $x = 10, 25$.

Обратными значениями (mod 35) и их квадратными корнями являются:

v	v^{-1}	s=sqrt(v^{-1})
1	1	1
4	9	3
9	4	2
11	16	4
16	11	9
29	29	8

Обратите внимание, что у чисел 14, 15, 21, 25 и 30 нет обратных значений mod 35, так как они не взаимно просты с 35. Это имеет смысл, так как должно быть (5 - 1) * (7 - 1)/4 квадратичных остатков mod 35, взаимно простых с 35: НОД(x, 35) = 1 (см. раздел 11.3).

Итак, Пегги получает открытый ключ, состоящий из k = 4 значений: {4,11,16,29}. Соответствующим закрытым ключом является {3,4,9,8}. Вот один этап протокола.

(1) Пегги выбирает случайное r=16, вычисляет 16^2 mod 35 = 11 и посылает его Виктору.

(2) Виктор посылает Пегги строку случайных битов: {1, 1, 0, 1}

(3) Пегги вычисляет $16*(3^1*4^1*9^0*8^1)$ mod 35 = 31 и посылает его Виктору.

(4) Виктор проверяет, что $31^2*(4^1*11^1*16^0*29^1)$ mod 35 =11.

Пегги и Виктор повторяют этот протокол t раз, каждый раз с новым случайным r, пока Виктор будет убежден.

Небольшие числа, подобные использованным в примере, не обеспечивают реальной безопасности . Но когда длина n равна 512 и более битам, Виктор не сможет узнать о закрытом ключе Пегги ничего кроме того факта, что Пегги действительно знает его.

Схема подписи Fiat-Shamir

Превращение этой схемы идентификации в схему подписи - это, по сути, вопрос превращения Виктора в хэш-функциию. Главным преимуществом схемы цифровой подписи Fiat-Shamir по сравнению с RSA является ее скорость: для Fiat-Shamir нужно всего лишь от 1 до 4 процентов модульных умножений, используемых в RSA. В этом протоколе снова вернемся к Алисе и Бобу.

Смысл переменных - такой же, как и в схеме идентификации. Выбирается n - произведение двух больших простых чисел. Генерируется открытый ключ, $v_1, v_2, \ldots v_k$, и закрытый ключ, $s_1, s_2, \ldots s_k$, где $s_i \equiv \text{sqrt}\ (v_i^{-1})$ (mod n).

(1) Алиса выбирает t случайных целых чисел в диапазоне от 1 до n - $r_1, r_2, \ldots, r_t$ - и вычисляет $x_1, x_2, \ldots x_t$, такие что $x_i = r_i^2$ mod n.

(2) Алиса хэширует объединение сообщения и строки x_i, создавая битовый поток: $H(m, x_1, x_2, \ldots x_t)$. Она использует первые $k*t$ битов этой строки в качестве значений b_{ij}, где i пробегает от1 до t, а j от 1 до k.

(3) Алиса вычисляет $y_1, y_2, \ldots y_t$, где $y_i = r_i *(s_1^{b_{i1}} * s_2^{b_{i2}} *\ldots* s_k^{b_{ik}})$ mod n

(Для каждого i она перемножает вместе значения s_i, в зависимости от случайных значений b_{ij}. Если b_{ij}=1, то s_i участвует в вычислениях, если b_{ij}=0, то нет.)

(4) Алиса посылает Бобу m, все биты b_{ij}, и все значения y_i. У Боба уже есть открытый ключ Алисы: $v_1, v_2, \ldots v_k$.

(5) Боб вычисляет $z_1, z_2, \ldots z_t$, где $z_i = y^2*(v_1^{b_{i1}} * v_2^{b_{i2}} *\ldots* v_k^{b_{ik}})$ mod n

(И снова Боб выполняет умножение в зависимости от значений b_{ij}.) Также обратите внимание, что z_i должно быть равно x_i.

(6) Боб проверяет, что первые $k*t$ битов $H(m, z_1, z_2, \ldots z_t)$ - это значения b_{ij}, которые прислала ему Алиса.

Как и в схеме идентификации безопасность схемы подписи пропорциональна $1/2^{kt}$. Она также зависит от сложности разложения n на множители. Фиат и Шамир показали, что подделка подписи облегчается, если сложность разложения n на множители заметно меньше 2^{kt}. Кроме того, из-за вскрытия методом дня рождения

З а д а н и е .

13. Ознакомиться с алгоритмом идентификации.
14. Разработать программу идентификации.
15. Произвести контрольную идентификацию.
16. Оценить затраченные ресурсы компьтера.
17. Оценить затраченные ресурсы и время.