Лабораторная работа № 6

Исследование потоковых шифров

Цель: изучение и приобретение практических навыков разработки и использования приложений для реализации потоковых шифров.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости потоковых шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для потокового зашифрования/расшифрования.
3. Выполнить анализ криптостойкости потоковых шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

6.1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

6.1.1 Классификация и общие свойства потоковых шифров

Потоковый шифр (иногда говорят «поточный») – симметричный шифр, преобразующий каждый символ m_i открытого текста в символ шифрованного, c_i, зависящий от ключа и расположения символа в тексте.

Термин «потоковый шифр» обычно используется в том случае, когда шифруемые символы открытого текста представляются одной буквой, битом или реже – байтом.

Все потоковые шифры делятся на 2 класса: *синхронные* и *асинхронные* (или *самосинхронизирующиеся*). Необходимые начальные сведения об общих характеристиках и свойствах потоковых шифров можно найти в гл. 6 из [2], а также в [4].

Основной задачей потоковых шифров является выработка некоторой последовательности (*гаммы*) для зашифрования, т.е. выходная гамма является ключевым потоком (ключом) для сообщения. В общем виде схема потокового шифра изображена на рис. 6.1.

Синхронные потоковые шифры (СПШ) характеризуются тем поток ключей генерируется независимо от открытого текста и шифртекста. Главное свойство СПШ – нераспространение ошибок. Ошибки отсутствуют, пока работают синхронно шифровальное и дешифровальное устройства отправителя и получателя информации. Один из методов борьбы с рассинхронизацией – разбить отрытый текст на отрезки, начало и конец которых выделить вставкой контрольных меток (специальных маркеров).

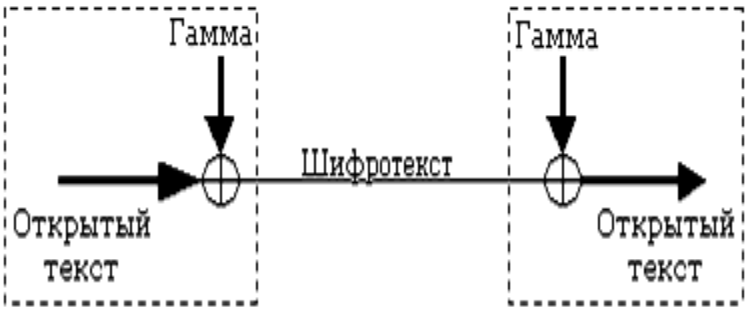

Рисунок 6.1 Схема потокового шифра

Синхронные потоковые шифры уязвимы к *атакам на основе изменения отдельных бит шифртекста*.

В *самосинхронизирующихся потоковых шифрах* символы ключевой гаммы зависят от исходного секретного ключа шифра и от конечного числа последних знаков зашифрованного текста. Основная идея заключается в том, что внутреннее состояние генератора потока ключей является функцией фиксированного числа предыдущих битов шифртекста. Поэтому генератор потока ключей на приемной стороне, приняв фиксированное число битов, автоматически синхронизируется с генератором гаммы.

Недостаток этих потоковых шифров – распространение ошибок, так как искажение одного бита в процессе передачи шифртекста приведет к искажению нескольких битов гаммы и, соответственно, расшифрованного сообщения.

6.1.2 Генераторы ключевой информации

Потоковый шифр максимально должен имитировать одноразовый блокнот. В соответствии с этим ключ должен по своим свойствам максимально походить на случайную числовую последовательность.

Ключевые последовательности (*случайные последовательности*, СП, либо *псевдослучайные последовательности*, ПСП) вырабатываются специальными блоками систем потокового шифрования – генераторами. В РБ в настоящее время действует стандарт СТБ 34.101.47-2017 "Информационные технологии и безопасность. Алгоритмы генерации псевдослучайных чисел" [31].

Стандарт устанавливает криптографические алгоритмы генерации псевдослучайных чисел. Алгоритмы стандарта могут применяться для построения ключей, синхропосылок, одноразовых паролей, других непредсказуемых или уникальных параметров криптографических алгоритмов и протоколов. Стандарт применяется при разработке, испытаниях и эксплуатации средств криптографической защиты информации.

Указанный стандарт определяет базовые понятия в рассматриваемой предметной области:

- *случайные числа* (последовательности) – последовательность элементов, каждый из которых не может быть предсказан (вычислен) только на основе знания предшествующих ему элементов данной последовательности;
- *псевдослучайные числа* – последовательность элементов, полученная в результате выполнения некоторого алгоритма и используемая в конкретном случае вместо последовательности случайных чисел.

6.1.2.1 Линейный конгруэнтный генератор

Часто используемый алгоритм генерирования (программно или аппаратно) ПСП реализуется на основе так называемого *линейного конгруэнтного генератора*, описываемого следующим рекуррентным соотношением:

$$x_{t+1} = (a*x_t + c) \bmod n, \qquad (6.1)$$

где: x_t и x_{t+1} – соответственно t-й (предыдущий) и (t+1)-й (текущий, вычисляемый) члены числовой последовательности; a, c и n – константы. Период такого генератора (период ПСП) не превышает n.

Если параметры a, b и c выбраны правильно, то генератор будет порождать случайные числа с максимальным периодом, равным c. При программной реализации значение c обычно устанавливается равным 2^{b-1} или 2^b, где b – длина слова в битах.

Достоинством линейных конгруэнтных генераторов псевдослучайных чисел является их простота и высокая скорость получения псевдослучайных значений. Линейные конгруэнтные генераторы находят применение при решении задач моделирования и математической статистики, однако в криптографических целях их нельзя рекомендовать к использованию, так как специалисты по

криптоанализу научились восстанавливать всю последовательность ПСЧ по нескольким ее значениям.

Генератор практически не используются в криптографии в силу низкой криптостойкости. Тем не менее, полезны для решения задач моделирования.

Комбинации нескольких (чаще двух) линейных конгруэнтных генераторов позволяют значительно повысить период ПСП. Б. Шнайер, например, приводит данные о том, как на 32-разрядных ПК реализовать генератор в виде комбинации двух, каждый из которых обеспечивает период соответственно $2^{31} - 85$ и $2^{31} - 249$, а комбинированный генератор позволяет достичь периода ПСП, равного произведению указанных чисел [4].

6.1.2.2 Генератор ПСП на основе регистров сдвига

Достаточно распространенным является использование *регистров сдвига* (РС) в качестве генераторов ПСП в силу простоты реализации на основе цифровой логики. РС с линейной обратной связью (РСЛОС) состоит из двух частей: собственно РС и функции обратной связи. На рис. 6.2 представлена общая схема РС с линейной обратной связью. Функция обратной связи реализуется с помощью сумматоров сложения по модулю два (элементы XOR; на рис. 6.2 обозначены в виде кружочков со знаком сложения).

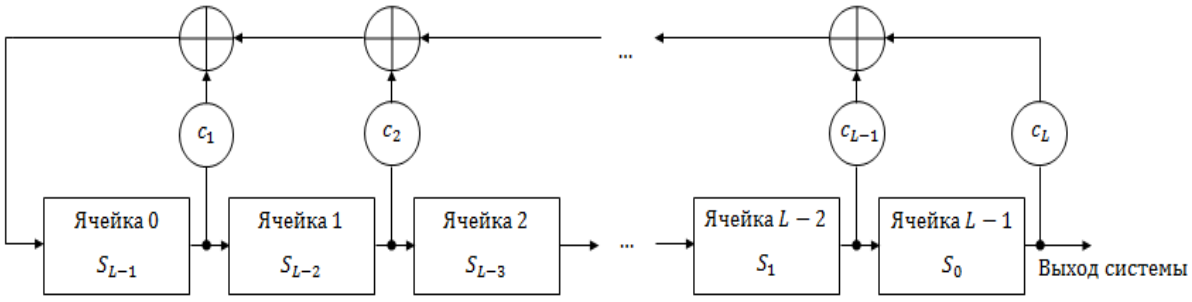

Рисунок 6.2 Общая схема регистра сдвига с линейной обратной связью

РСЛОС строятся на основе *примитивных порождающих полиномов* (*многочленов*), которые мы подробно анализировали при изучении циклических помехоустойчивых кодов [1] (см. также [32, 33]). Если многочлен является неприводимым, то период ПСП при ненулевом начальном условии (ненулевом состоянии) регистра будет максимально возможным: $2^L - 1$.

6.1.2.3 Генератор псевдослучайных чисел на основе алгоритма RSA

Собственно алгоритм RSA разработан для систем асимметричного зашифрования/расшифрования и будет более детально рассмотрен с практической точки зрения ниже.

Генератор же ПСП на основе RSA устроен следующим образом. Последовательность генерируется с использованием соотношения

$$x_t = (x_{t-1})^e \mod n. \qquad (6.2)$$

Начальными параметрами служат n, большие простые числа p и q (причем $n = p*q$), целое число e, взаимно простое с произведением (p – 1)*(q –1), а также некоторое случайное начальное значение, x_0.

Выходом генератора является на t-м шаге является младший бит числа x_t.

Безопасность генератора опирается на сложности взлома алгоритма RSA, т. е. на разложении числа n на простые сомножители.

6.1.2.4 Генератор псевдослучайных чисел на основе алгоритма BBS

Широкое распространение получил алгоритм генерации ПСП, называемый алгоритмом BBS (от фамилий авторов: L. Blum, M. Blum, M. Shub) или *генератором на основе квадратичных вычетов*. Для целей криптографии этот метод предложен в 1986 г.

Начальное значение x_0 генератора вычисляется на основе соотношения

$$x_0 = x^2 \mod n, \qquad (6.3)$$

где n, как и в генераторе на основе RSA, является произведением простых чисел p и q, однако в нашем случае эти простые числа должны быть сравнимы с числом 3 по модулю 4, т. е. при делении p и q на 4 должен получаться одинаковый остаток: 3; число x должно быть взаимно простым с n; число n называют *числом Блюма*.

Выходом генератора на t-м шаге является младший бит числа x_t:

$$x_t = (x_{t-1})^2 \mod n. \qquad (6.4)$$

Рассмотрим пример.

Пример. Пусть $p = 11$, $q = 19$ (убеждаемся, что 11 mod 4 = 3, 19 mod 4 = 3). Тогда $n = p* q = 11*19=209$. Выберем x, взаимно простое с n: пусть $x = 3$ [34].

Вычислим стартовое число генератора x_0 в соответствии с (6.3):

$$x_0 = x^2 \bmod n = 3^2 \bmod 209 = 9 \bmod 209 = 9.$$

Вычислим первые десять чисел x_t по алгоритму BBS.

В качестве случайных бит будем брать младший бит в двоичной записи числа x_t (табл. 6.1).

Таблица 6.1

Пояснение к методу генерации ПСП на основе метода BBS

Число x_t	Младший бит двоичного числа x_t
$x_1 = 9^2 \bmod 209 = 81 \bmod 209 = 81$	1
$x_2 = 81^2 \bmod 209 = 6561 \bmod 209 = 82$	0
$x_3 = 822 \bmod 209 = 6724 \bmod 209 = 36$	0
$x_4 = 36^2 \bmod 209 = 1296 \bmod 209 = 42$	0
$x_5 = 42^2 \bmod 209 = 1764 \bmod 209 = 92$	0
$x_6 = 92^2 \bmod 209 = 8464 \bmod 209 = 104$	0
$x_7 = 104^2 \bmod 209 = 10816 \bmod 209 = 157$	1
$x_8 = 157^2 \bmod 209 = 24649 \bmod 209 = 196$	0
$x_9 = 196^2 \bmod 209 = 38416 \bmod 209 = 169$	1
$x_{10} = 169^2 \bmod 209 = 28561 \bmod 209 = 137$	1

С точки зрения безопасности важным является свойство рассмотренного генератора, заключающееся в том, что при известных p и q x_t-й бит легко вычисляется без учета предыдущего (x_{t-1}) бита:

$$x_t = (x_0)^a \bmod n, \qquad (6.4)$$

где $a = 2^t \bmod ((p-1)(q-1))$.

Алгоритм является сравнительно медленном. Для ускорения можно использовать не последний бит числа, а несколько последних бит. Однако, понятно, что при этом алгоритм является менее криптостойким.

6.1.3 Потоковый шифр RC4

Алгоритм RC4 разработан Р. Ривестом в 1987 г. Представляет собой потоковый шифр с переменным размером ключа. Здесь гамма не зависит от открытого текста [4].

Алгоритм RC4, как и любой потоковый шифр, строится на основе генератора псевдослучайных битов (генератора ПСП). На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Длина ключа может составлять от 40 до 2048 бит.

Ядро алгоритма состоит из функции генерации ключевого потока. Другая часть алгоритма – функция инициализации, которая использует *ключ переменной длины* K_i для создания начального состояния генератора ключевого потока.

В основе алгоритма – размер блока или слова, определяемый параметром *n*. Обычно $n = 8$, но можно использовать и другие значения. Внутренне состояние шифра определяется массивом слов (*S*-блоком) размером 2^n. При $n = 8$ элементы блока представляют собой перестановку чисел от 0 до 255, а сама перестановка зависит от ключа переменной длины. Другими элементами внутреннего состояния являются 2 счетчика (каждый размером в одно слово; обозначим их i и *j*) с нулевыми начальными значениями. В основе вычислений лежит операция по *mod* 2^n.

Генератор ключевого потока RC4 переставляет значения, хранящиеся в *S,* и каждый раз выбирает различное значение из *S* в качестве результата. В одном цикле RC4 определяется одно *n*-битное слово *K* из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста. Эта часть алгоритма называется генератором ПСП. При n = 8 для генерации случайного байта выполняются операции, представленные листингом 6.1.

```
i = (i + 1) mod 256;
j = (j + Si) mod 256;
поменять местами Si и Sj;
a = (Si + Sj) mod 256;
K = Sa
```

Листинг 6.1 Псевдокод для генерации байта ПСП

Байт *K* используется в операции XOR с открытым текстом для получения 8-битного шифртекста или для его расшифрования.

Так же достаточно проста и инициализация *S*-блока. Этот алгоритм использует ключ, который подается на вход пользователем. Сначала *S*-блок заполняется линейно: $S_0 = 0$, $S_1 = 1$, …, $S_{255} = 255$. Затем заполняется секретным ключом другой 256-байтный массив. Если необходимо, ключ повторяется многократно, чтобы заполнить весь массив: K_0, K_1, …, K_{255}. Далее массив *S* перемешивается путем перестановок, определяемых ключом. Действия выполняются в соответствии с псевдокодом, представленным листингом 6.2.

```
1. j = 0; i =0;
2. j = (j + Si + Ki) mod 256;
3. поменять местами Si и Sj;
4. i = i +1;
5. если i <256, то перейти на п.2
```

Листинг 6.2 Псевдокод для начального заполнения таблицы замен S (S-блока)

Проиллюстрируем работу алгоритма для случая $n = 4$, воспользовавшись примером из [35].

Пример. Предположим, что секретный ключ состоит из шести 4-битных значений (приведем их в десятичном виде): 1, 2, 3, 4, 5, 6. Для его получения сгенерируем последовательность чисел, для чего заполним таблицу S линейно (последовательно) 16 (2^4) числами от 0 до 15 в соответствии с табл. 6.2.

Таблица 6.2

Начальное заполнение таблицы S

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

Подготовим таблицу K, записав в нее ключ необходимое количество раз (табл. 6.3).

Таблица 6.3

Заполнение таблицы K

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

Перемешаем содержимое таблицы S. Для этого будем использовать алгоритм в соответствии с листингом 6.2 (для $n = 4$). Процесс выполнения представим в виде трассировочной таблицы (табл. 6.4).

Таблица 6.4

Подготовительный этап (инициализация таблицы замен) алгоритма RC4

Номер пункта алг.	Выполняемое действие (по **mod 16**)	Новое значение *i*	Новое значение *j*
1	j = 0; i =0	0	
2	$j = j + S_i + K_i = 0 + 0 + 1 = 1$		1
3	Поменять местами S_i и S_j, т. е. S_0 и S_1		

4	i = i +1	1	
5	i < 16, поэтому перейти на п.2		
2	j = j + S_i + K_i = 1 + 0+ 2 = 3		3
3	Поменять местами S_i и S_j, т. е. S_1 и S_3		
4	i = i +1	2	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (3 + 2 + 3) mod 16 = 8		8
3	Поменять местами S_i и S_j, т. е. S_2 и S_8		
4	i = i +1	3	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (8 + 0 + 4) mod 16 = 12		12
3	Поменять местами S_i и S_j, т. е. S_3 и S_{12}		
4	i = i +1	4	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (12 + 4 + 5) mod 16 = 5		5
3	Поменять местами S_i и S_j, т. е. S_4 и S_5		
4	i = i +1	5	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (5 + 4 + 6) mod 16 = 15		15
3	Поменять местами S_i и S_j, т. е. S_5 и S_{15}		
4	i = i +1	6	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (15 + 6 + 1) mod 16 = 6		6
3	Поменять местами S_i и S_j, т. е. S_6 и S_6		
4	i = i +1	7	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (6 + 7 + 2) mod 16 = 15		15
3	Поменять местами S_i и S_j, т. е. S_7 и S_{15}		
4	i = i +1	8	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (15 + 2 + 3) mod 16 = 4		4
3	Поменять местами S_i и S_j, т. е. S_8 и S_4		
4	i = i +1	9	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (4 + 9 + 4) mod 16 = 1		1
3	Поменять местами S_i и S_j, т. е. S_9 и S_1		
4	i = i +1	10	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (1 + 10 + 5) mod 16 = 0		0
3	Поменять местами S_i и S_j, т. е. S_{10} и S_0		
4	i = i +1	11	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (0 + 11 + 6) mod 16 = 1		1
3	Поменять местами S_i и S_j, т. е. S_{11} и S_1		

4	i = i +1	12	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (1 + 0 + 1) mod 16 = 2		2
3	Поменять местами S_i и S_j, т. е. S_{12} и S_2		
4	i = i +1	13	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (2 + 13 + 2) mod 16 = 1		1
3	Поменять местами S_i и S_j, т. е. S13 и S1		
4	i = i +1	14	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (1 + 14 + 3) mod 16 = 2		2
3	Поменять местами S_i и S_j, т. е. S_{14} и S_2		
4	i = i +1	15	
5	i < 16 , поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (2 + 7 + 4) mod 16 = 13		13
3	Поменять местами S_i и S_j, т. е. S_{15} и S_{13}		
4	i = i +1	16	
5	i < 16 – неверно, поэтому закончить		

После этого получим инициализированную и подготовленную к основному этапу таблицу S (табл. 6.5).

Таблица 6.5

Таблица S

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

Далее выполняем генерацию случайных 4-битных слов. Вычислим первые 5 чисел псевдослучайной последовательности, используя алгоритм в листинге 6.1 (для n = 4). Результаты вычисления последовательности значений также представим в виде таблицы (табл. 6.6).

Таблица 6.6

Вычисление элементов ПСП (K_i) на основе алгоритма RC4

Вычисление K_i	Выполняемое действие (по mod 16)	Новое знач. i	Новое знач. j	Новое знач. a
K_1	1. i = (i + 1) =0+1=1	1		
	2. j = (j + S_i) mod 16=(0+13) mod 16=13		13	
	3. Поменять местами S_1 и S_{13}			

	4. a = (S_i + S_j)mod 16=(7+13)mod 16=4			4
	5. **K_1 = S_4 =2**			
K_2	1. i = (i + 1) =1+1=2	2		
	2. j = (j + S_i) mod 16=(13+14) mod 16=11		11	
	3. Поменять местами S_2 и S_{11}			
	4. a = (S_i + S_j)mod 16=(9+14)mod 16=7			7
	5. **K_2 = S_7=4**			
K_3	1. i = (i + 1) =2+1=3	3		
	2. j = (j + S_i) mod 16=(11+12) mod 16=7		7	
	3. Поменять местами S_3 и S_7			
	4. a = (S_i + S_j)mod 16=(4+12)mod 16=0			0
	5. **K_3 = S_0=10**			
K_4	1. i = (i + 1) =3+1=4	4		
	2. j = (j + S_i) mod 16=(7+2) mod 16=9		9	
	3. Поменять местами S_4 и S_9			
	4. a = (S_i+ S_j)mod 16=(3+2)mod 16=5			5
	5. **K_4= S_5=15**			
K_5	1. i = (i + 1) =4+1=5	5		
	2. j = (j + S_i) mod 16=(9+15) mod 16=8		8	
	3. Поменять местами S_5 и S_8			
	4. a = (S_i+ S_j)mod 16=(5+15)mod 16=4			4
	5. **K_5= S_4 =3**			

В результате первые пять значений гаммы получились следующие: 2, 4, 10, 15, 3. При необходимости получения большего количества случайных чисел можно продолжить вычисления дальше. При n=4 генерируемые числа будут иметь размер 4 бита, т. е. для нашего примера: 0010, 0100, 1010, 1111, 0011. Примеры программной реализации RC4 можно найти, в частности, в [36].

По утверждению [4] рассмотренный алгоритм устойчив к линейному и дифференциальному криптоанализу. Кроме того, при n = 8 этот алгоритм может находиться примерно в 2^{1700} состояниях (256!*256^2).

При n = 16 алгоритм должен обладать большей в сравнении с n = 8 скоростью, но при этом начальные установки занимают гораздо больше времени – нужно заполнить 65536-элементный массив.

Шифр RC4 применяется в некоторых широко распространённых стандартах и протоколах шифрования таких, как WEP, WPA и TLS, а также в Kerberos и др..

Реализация алгоритма на Python приведена в листинге 6.3.

```
## rc4.py - stream cipher RC4

def RC4crypt(data, key):
    """RC4 algorithm"""
    x = 0
    box = range(256)
    for i in range(256):
        x = (x + box[i] + ord(key[i % len(key)])) % 256
        box[i], box[x] = box[x], box[i]
    x = y = 0
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
        out.append(chr(ord(char) ^ box[(box[x] + box[y]) % 256]))

    return ''.join(out)

def hexRC4crypt(data, key):
    """hex RC4 algorithm"""
    dig = crypt(data, key)
    tempstr = ''
    for d in dig:
        xxx = '%02x' % (ord(d))
        tempstr = tempstr + xxx

    return tempstr

assert hexRC4crypt('The quick brown fox jumps over the lazy dog',
'123456') == \

'54901b4755467cff141740c496492fee8ba7224a99f52cc2e84d019e1c0cda32b6a
96d521d067d00ce6716'
assert
RC4crypt('\xA4\x8F\x9B\xFF\x6D\x3A\xFD\x7D\x56\xCB\xAC\xD6\x46\x27\x
50\x65', '10001') == \
  'Wow, you did it!'
```

Листинг 6.3 Реализация RC4 на Python

Другими известными потоковыми шифрами являются, например, SEAL, программный код которого приведен в [4], и WAKE.

Наилучшие характеристики будут иметь генераторы случайных чисел, основанные на «естественных случайностях», свойственных, например, процессам в радиоэлектронной аппаратуре, в системах телекоммуникаций, в приемах работы с клавиатурой операторов и др.

6.2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1. Разработать авторские многооконные приложения в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими заданные алгоритмы.

Приложение 1 должно реализовывать генерацию ПСП в соответствии с вариантом из табл. 6.6.

Таблица 6.6

Вариант задания	Алгоритм генерации ПСП	Параметры
1	RSA	p, q, e – 512-разрядные числа (обосновать выбор)*
2	RSA	p, q, e – 256-разрядные числа (обосновать выбор)*
3	BBS	n = 256; p, q – обосновать выбор по согласованию с преподавателем
4	BBS	n = 512; p, q – обосновать выбор по согласованию с преподавателем
5	Линейный конгруэнтный генератор	a = 421, c = 1663, n = 7875
6	Линейный конгруэнтный генератор	a = 430, c = 2531, n = 11979

* – можно воспользоваться результатами выполнения ЛР №1

Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.7, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.

Таблица 6.7

Вариант задания	n	Ключ (в виде десятичных чисел)
1	8	121, 14, 89, 15
2	8	76, 111, 85, 54, 211
3	8	43, 45, 100, 21, 1
4	8	12, 13, 90, 91, 240

5	8	123, 125, 41, 84, 203
6	6	1, 11, 21, 31, 41, 51
7	6	10, 11, 12, 13, 14, 15
8	6	20, 21, 22, 23, 60, 61
9	6	61, 60, 23, 22, 21, 20
10	6	15, 14, 13, 12, 11, 10
11	8	13, 19, 90, 92, 240
12	8	122, 125, 48, 84, 201
13	8	61, 60, 23, 22, 21, 20
14	8	20, 21, 22, 23, 60, 61
15	8	1, 11, 21, 31, 41, 51

В качестве шифруемого сообщения может быть выбран произвольный текст
2. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. В чем состоит особенность потоковых шифров?
2. В чем состоят преимущества и недостатки синхронных и асинхронных потоковых шифров?
3. Какими свойствами должен обладать генератор псевдослучайных чисел для использования в криптографических целях?
4. Дать характеристику линейным конгруэнтным генераторам. Области их применения.
5. Значения x_0, x_1, x_2, x_3, полученные с помощью линейного конгруэнтного генератора, равны соответственно: 1, 12, 3, 6. Найти параметры a, c и n генератора ПСЧ, удовлетворяющие (6.1).
6. Представить общую структурную схему генератора ПСП на основе регистров сдвига с линейной обратной связью. Пояснить особенности его функционирования.
7. Синтезировать структурную схему генератора ПСП на основе регистров сдвига с линейной обратной связью, формально обозначаемого следующим образом: а) 3210, б) 420, в) 5410, г) 520, д) 84320. Составить таблицу состояний генератора и определить период ПСП.
8. Определить первые 12 бит ПСП, задаваемого формально в виде чисел 5410, если начальные состояния ячеек (слева-направо) соответствуют последовательности 10101.

9. Как устроен генератор ПСП на основе RSA? На чем основана криптостойкость реализуемого алгоритма?
10. Вычислить x_1, x_5 , x_9, x_{11} по методу генерации псевдослучайных чисел BBS, если $p = 11$, $q = 19$, $x = 3$.
11. Пояснить базовый алгоритм, реализованный в шифре RC4.
12. Пояснить принципы формирования истинных случайных последовательностей, основанных на «естественных случайностях».