МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Криптографические методы защиты информации

Электронный учебно-методический комплекс
по дисциплине в LMS Moodle

Работа выполнена по мероприятию блока 1 «Совершенствование образовательной деятельности» Программы развития СГАУ на 2009 – 2018 годы по проекту «Разработка контента для системы электронного и дистанционного обучения по основным образовательным программам факультета информатики»
Соглашение № 1/34 от 3.06.2013 г.

САМАРА
2013

УДК 004.056
К 82

Автор-составитель: **Митекин Виталий Анатольевич**

Криптографические методы защиты информации [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle / Мин-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост. В.А.Митекин. - Электрон. текстовые и граф. дан. - Самара, 2013. – 1 эл. опт. диск (CD-ROM).

В состав учебно-методического комплекса входят:

1. Курс лекций.
2. Методические указания к лабораторным работам.
3. Тест по курсу лекций.
4. Рабочая программа

УМКД «Криптографические методы защиты информации» предназначен для студентов факультета информатики, обучающихся по направлению подготовки специалистов 090303 «Информационная безопасность автоматизированных систем» в 8 семестре.

УМКД разработан на кафедре Геоинформатики и информационной безопасности.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Кафедра геоинформатики и информационной безопасности

КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Конспект лекций

Самара 2013

ЛЕКЦИЯ 1. ОСНОВНЫЕ ЦЕЛИ И ПОНЯТИЯ КРИПТОГРАФИИ. КЛАССИФИКАЦИЯ ШИФРОВ

1.1 Основные задачи криптографии

В переводе с греческого языка слово криптография означает тайнопись. Смысл этого термина выражает основное предназначение криптографии – сохранить в тайне необходимую информацию. Современная ***криптография*** является областью знаний, связанной с решением таких проблем безопасности информации, как конфиденциальность, целостность, аутентификация сообщений, а также обеспечение невозможность отказа сторон от авторства сообщения. Достижение этих требований безопасности информационного воздействия и составляет основные цели криптографии. Они определяются следующим образом:

1) обеспечение ***конфиденциальности*** – решение проблемы защиты информации от ознакомления с ее содержанием со стороны лиц, не имеющих права доступа к ней;
2) обеспечение ***целостности*** – гарантированное обнаружение любого факта несанкционированного изменения информации, в том числе в процессе передачи данной информации по каналам связи;
3) обеспечение ***аутентификации*** – разработка методов подтверждения подлинности сторон и самой информации в процессе информационного взаимодействия;
4) обеспечение ***невозможности отказа от авторства*** – предотвращение возможности отказа субъектов от переданных ими сообщений.

Рассмотрим средства для достижения этих целей более подробно.

В простейшем случае задача по обеспечению конфиденциальности информации описывается взаимодействием трех субъектов (сторон). Владелец информации (***отправитель***), осуществляет преобразование исходной (***открытой***) информации в форму передаваемых ***получателю*** по открытому каналу связи с целью ее защиты от противника (атакующего).

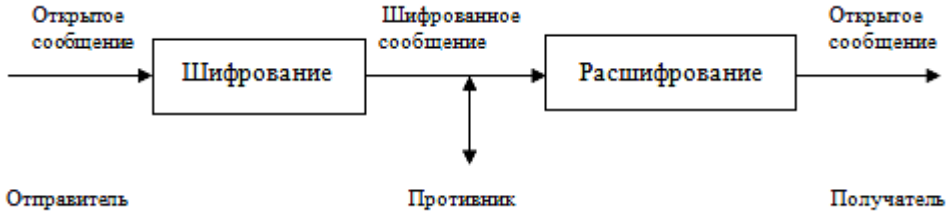


Рисунок 1. Передача шифрованной информации

Действия атакующего по захвату защищаемой информации называют ***атаками***. Атаки делятся на активные и пассивные. ***Пассивные*** атаки связаны с прослушиванием, анализом трафика, перехватом, дешифрованием, т.е. попытками «взломать» защиту с целью овладения информацией. При проведении ***активных*** атак противник может прерывать процесс передачи сообщений, создавать поддельные или модифицировать передаваемые шифрованные сообщения.

Под ***шифром*** понимается семейство обратимых преобразований, каждое из которых определяется некоторым секретным параметром (ключом) а также порядком применения данного преобразования, называемым ***режимом шифрования***.

Подчеркнем разницу между терминами «***расшифрование***» и «***дешифрование***». При расшифровании действующий ключ считается известным, в то время как при дешифровании ключ неизвестен. Для разных шифров задача дешифрования имеет различную сложность. Уровень сложности этой задачи и определяет главное свойство шифра – способность противостоять попыткам атакующего завладеть защищаемой информацией. В связи с этим говорят о ***криптографической стойкости шифра*** (или просто стойкости), различая более и менее стойкие шифры.

Ключ – это важнейший компонент шифра, отвечающий за выбор преобразования, применяемого для зашифрования конкретного сообщения. Традиционно в качестве ключа использовалась короткая буквенная или числовая последовательность (пароль). В настоящее время большинство применяемых криптографических алгоритмов в качестве ключа используют последовательность бит фиксированной длины (обычно от 40 до 4000 бит).

Чтобы однозначно задать преобразование ***открытого сообщения*** в ***зашифрованное сообщение***, необходимо и достатоочно определить ***ключ*** и ***криптографический алгоритм (алгоритмы зашифрования и расшифрования)***. Пару алгоритмов зашифрования и расшифрования называют ***криптосистемой*** (шифросистемой), а реализующие их устройства – ***шифротехникой***.

Если обозначить через M открытое, а через C шифрованное сообщения, то процессы зашифрования и расшифрования можно записать в виде равенств

$$E_{K_1}(M) = C, \qquad (1)$$
$$D_{K_2}(C) = M, \qquad (2)$$

в которых алгоритмы зашифрования E и расшифрования D должны удовлетворять равенству

$$D_{K_2}\left(E_{K_1}(M)\right) = M. \quad (3)$$

Фактически, выражение (3) определяет требование однозначности преобразования открытого сообщения в зашифрованное (и наоборот). Таким обраазом, при известном криптографическом алгоритме и ключе любому открытому сообщению M может быть поставлено в соответсвие только одно зашифрованное сообщение $\mathrm{C} = \mathrm{E}_{\mathrm{K}_1}(\mathrm{M})$

Различают ***симметричные*** и ***ассиметричные*** криптосистемы. В симметричных системах знание ключа зашифрования K_1 позволяет легко найти ключ расшифрования K_2 (в большинстве случаев K_1=K_2). В асимметричных криптосистемах ключи K_1 *и* K_2 различны, и в большинстве случаев знание ключа K_1 не позволяет определить ключ K_2.

Таким образом, для симметричных криптосистем ключ необходимо сохранять в тайне и обеспечивать его безопасное согласование (передачу получателю зашифрованного сообщения). Для асимметричных систем ключ K_2 сохраняется в тайне, ключ K_1 можно сделать общедоступным. В связи с

этим асимметричные криптосистемы называют еще ***шифрами с открытым ключом***.

Наряду с конфиденциальностью не менее важной задачей является обеспечение целостности информации, другими словами, - неизменности ее в процессе передачи или хранении. Решение этой задачи предполагает разработку средств, позволяющих обнаруживать не столько случайные искажения, сколько целенаправленное навязывание атакующим ложной информации. Для этого в передаваемую информацию вносится ***избыточность***, то есть к сообщению добавляется некоторая проверочная информация, которая играет роль «контрольной суммы» для проверки целостности полученного сообщения. Алгоритм выработки проверочной комбинации является «криптографическим», то есть зависящим от секретного ключа, что не позволяет атакующему сформировать корректную проверочную информацию и, таким образом, должно исключить успешное навязывание атакующим искаженной или ложной информации.

Для проверки целостности обычно к сообщению *M* добавляется проверочная комбинация *S*, называемая ***кодом аутентификации сообщения*** (КАС, MAC в англоязычной литературе) или ***имитовставкой***. В этом случаи по каналу связи передается пара *C=(M,S)*. При получении сообщения *M* пользователь вычисляет значение проверочной комбинации от этого сообщения и сравнивает его с полученным контрольным значением *S*. Несовпадение означает то, что данные были изменены.

Аутентификация – установление подлинности. Установление подлинности всех аспектов информационного взаимодействия (сеанс связи, личность передающей и принимающей сторон, передаваемые сообщения и др.) является важной составной частью проблемы обеспечения достоверности. Применительно к сеансу связи аутентификация означает проверку: целостности соединения, обеспечение невозможности несанкционированной повторной передачи данных атакующим, а также своевременность передачи данных. Один из способов – вставка в сообщение специальных чисел или ***меток времени***. Применительно к сторонам взаимодействия аутентификация означает проверку одной из сторон на соответствие заявленной, а предшествует ей процедура ***идентификации*** – процедура установления присвоенного данной стороне уникального системного имени-идентификатора. Основным средством для проведения идентификации являются ***протоколы идентификации***, позволяющие осуществлять идентификацию (и аутентификацию) каждой из участвующих во взаимодействии и не доверяющих друг другу сторон. Различают ***протоколы односторонней*** и ***взаимной идентификации***. ***Протокол*** – это распределенный алгоритм, определяющий последовательность действий каждой из сторон, а также порядок и формат передачи данных между сторонами. Наконец, применительно к самой информации аутентификация означает проверку того, что информация, передаваемая по каналу, является подлинной по содержанию, источнику, времени создания, времени пересылки и т.д. Проверка подлинности содержания информации сводится, по сути, к проверке ее неизменности (с момента ее создания) в процессе передачи или хранения, то есть проверке целостности. ***Аутентификация источника***

данных означает подтверждение того, что исходный документ был создан именно заявленным источником.

Появление ***цифровой подписи*** как средства аутентификации было обусловлено несколькими причинами:

- отправитель мог отказаться от факта передачи сообщения, утверждая, что его создал получатель (***отказ от авторства***);
- получатель мог модифицировать сообщение, а затем утверждать, что оно получено от отправителя именно в таком, модифицированном (***приписывание авторства***).

Оба этих недостатка связаны с применением симметричных криптографических алгоритмов, т.е. алгоритмов, предполагающих наличие общего секретного ключа и у отправителя, и у получателя.

Схема цифровой подписи включает два алгоритма, один – для вычисления, а второй – для проверки подписи. Одновременно с проблемой цифровой подписи возникла проблема построения бесключевых криптографических ***хэш-функций***. При вычислении цифровой подписи оказывается, что более удобным осуществлять хэширование, то есть свертку текста в некоторую комбинацию фиксированной длины, а затем уже подписывать полученную комбинацию с помощью секретного ключа. При этом функция хэширования, хотя и не зависит от ключа и является открытой должна быть «криптографической», то есть обладать свойством односторонности: по значению комбинации-свертки никто не должен иметь возможность подобрать соответствующее сообщение.

Порядок использования криптографической системы определяется ***системами установки*** и ***управления ключами***. Система установки ключей определяет алгоритмы и процедуры генерации, распределения, передачи и проверки ключей. Система управления ключами определяет порядок использования, смены, хранения и архивирования, резервного копирования и восстановления, замены или изъятия из обращения скомпрометированных, а также старых ключей.

1.2 Классификация шифров

Первичный признак, по которому производится классификация шифров, используется тип преобразования, осуществляемого с открытым текстом при шифровании.

1) ***Шифры подстановки***: фрагменты открытого сообщения (буквы, биты или слова) при шифровании заменяются символами того же или другого алфавита в соответствии с заранее обусловленной схемой замены;
2) ***Шифры перестановки***: символы шифруемого текста переставляются по определенному правилу в пределах блока текста;
3) ***Композиционные шифры***: шифрованный текст, полученный применением некоторого шифра, может быть еще раз зашифрован с помощью другого шифра.

Шифры подстановки

Обобщенно любой шифр подстановки можно разделить на два этапа:

1) разбиение открытого сообщения (текста, последовательности бит) на фрагменты фиксированной длины, которые будут шифроваться независимо (например, шифр подстановки может разбивать сообщения на отдельные буквы, слова, или на группы по 5 бит);
2) шифрование в соответствии с некоторым правилом замены, определяющим как фрагменты исходного текста заменяются на фрагменты зашифрованного текста.

Шифр Цезаря

Является частным случаем шифра простой подстановки (***одноалфавитной подстановки***). Заменяющая буква определяется путем смещения по алфавиту от исходной буквы на *К* букв. При достижении конца алфавита выполняется циклический переход к его началу. Рассмотрим на примере слова «*ПРИВЕТ*», со смещением *К* = 4:

«*ПРИВЕТ*», тогда *П* → *У*, *Р* → *Ф*, *И* → *М*, *В* → *Е*, *Е* → *И*, *Т* → *Ц*;
получаем шифротекст «*УФМЕИЦ*».

Достоинством системы шифрования Цезаря является простота зашифрования и расшифрования.

К недостаткам системы Цезаря следует отнести следующие:

- подстановки, выполняемые в соответствии с системой Цезаря, не маскируют частот появления различных букв исходного открытого текста;
- число возможных ***различных*** ключей *К* мало;

Шифр Виженера

Шифр Виженера относится к простой форме многоалфавитной подстановки.

Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Приведенный ниже пример показывает исходный текст и соответствующий ему зашифрованный текст. Ключом в данном примере является последовательность чисел (значения сдвигов) 1,2,2,1

Рисунок 2.

Шифр Плейфера

Если при шифровании две буквы открытого текста преобразуются в другую форму, то такой шифр называется биграммным шифром замены. Разобьем на пары букв исходный текст:

|*ПР*| *ИВ*| *ЕТ*| |*ПЕ*| *ТР*|,

Далее, в качестве ключа шифрования в шифре используется прямоугольная таблица, заполненная в некотором псевдослучайном порядке символами алфавита, и, в некоторых вариациях шифра, основными знаками препинания. Каждая биграммная (двухбуквенная) комбинация шифруется независимо от другой, по следующим правилам:

- если буквы пары не лежат в одной строке или в одном столбце таблицы-ключа, то они заменяются буквами, образующими с исходными буквами вершины прямоугольника. Первой букве пары соответствует буква таблицы, находящаяся в том же столбце;
- если буквы пары открытого текста расположены в одной строке таблицы, то каждая буква заменяется соседней справа буквой таблицы;
- если буквы пары лежат в одном столбце, то каждая буква заменяется соседней буквой снизу;
- если буквы в паре одинаковые, то между ними вставляется определенная буква, называемая «буквой-пустышкой». После этого разбиение на пары производится заново.

Табличная биграммная подстановка.

Более общим вариантом шифра Плейфера является шифр табличной биграммной подстановки. В данном случае ключом шифрования выступает таблица, в которой записаны все двухбуквенные комбинации:

|*АА*|*АБ*| *АВ* |…..|*ЯЭ*|*ЯЮ*|*ЯЯ*|

↓ ↓ ↓

|*ТУ*|*КЛ*|*МЖ*| - все сочетания из 2-х букв.

При шифровании открытого сообщения очередная биграммная комбинация заменяется на биграммную комбинацию, стоящую под ней в таблице подстановок (см. выше).

В результате подсчета всех парных комбинаций получим 33^2, т.е. таблица-ключ должна содержать более 1000 комбинаций. Количество различных ключей при этом будет равно числу возможных перестановок из 33^2, элементов:$(33^2)!$. Стойкость такого ключа к подбору значительно выше, чем у шифра Плейфера. На практике биграммную подстановку, проводимую «вручную», удобнее проводить используя две таблицы подстановок – отдельно для зашифрования и расшифрования. Пример таких таблиц и принцип их заполнения приведен на рис. 3.

При расшифровывании применяется обратный порядок действий.

$ET \rightarrow MM$

	А	Б	В	Г	...	Т
А						
Б						
В						
Г		ТМ				
...						
...						
Е						ММ

	А	Б	В	Г	...	М
А						
Б						
В						
Г						
...						
...						
М						ЕТ

Рисунок 3. Таблица биграммной подстановки и таблица рашифрования биграммной подстановки

Достоинством системы шифрования Плейфеера:

- шифрование биграммами резко повышает стойкость шифров к «ручному» вскрытию ,вскрытие с использованием статистических характеристик естественного языка требует гораздо большего объема зашифрованных сообщений.

К недостаткам системы Плейфеера можно отнести следующие:

- несложно определить ключ, если известны пара: зашифрованное сообщение+открытое сообщение.

Шифр перестановки

Алгоритм перестановки для всех шифров этого класса:

1) разбиение исходного текста на элементы;
2) смена порядка следования символов исходного текста.

Блочная перестановка

Схемы перестановок рассмотрим на исходном тексте - «*ПРИВЕТ*»

J_n=3 – разбиваем текст на группы по три символа, тем самым получаем:

ПРИ| *ВЕТ*, придумываем ключ, перемешиваем числа от 1 до *n* (например: 312).

3 1 2 3 1 2

|*П Р И*| *В Е Т*|, затем осуществляем внутри блока перестановку:

|*Р И П*| *Е Т В*|.

Достоинства данного типа перестановки:

- метод анализа частот не работает, так как частоты появления букв совпадут для зашифрованного и открытого сообщения;
- простота реализации.

Решетка Кардано

Приспособление: решетка из непрозрачного материала, в которой вырезаны окошки в «произвольном» порядке.

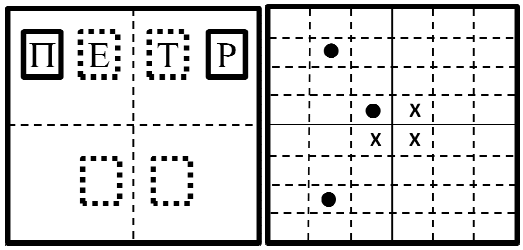


Рисунок 4. Решетка Кардано

Решетка помещается на лист бумаги и в прямоугольных отверстиях пишется сообщение, в каждой помещается отдельный символ, слог или целое слово. Исходное сообщение оказывается разделённым на большое число маленьких фрагментов. Затем решётка убирается и пустые места на бумаге заполняются посторонним текстом так, чтобы скрываемый текст стал частью шифртекста.

Лекция 2. Постулаты Керкхоффа. Понятие идеального шифра по Шеннону. Практическая стойкость шифра

2.1 Постулат Керкхоффа

Постулат Керкхоффа: устойчивость шифра к взлому должна обеспечиваться исходя из предположения, что взломщику известны все детали шифра, кроме значения ключа (язык сообщения, способ шифрования, длина сообщения и др.).

Следствия:

1) разглашение алгоритма не облегчает дешифрование, если ключ остается неизвестен;
2) В секрете необходимо хранить только ключ шифрования, алгоритм можно и нужно публиковать открыто (что позволяет провести его всесторонний анализ).

2.2 Расстояние единственности, способы дешифрования сообщений на естественном языке

Шеннон ввел в криптографию понятие ***избыточности естественного языка*** (т.е. осмысленных слов в языке гораздо меньше, чем произвольных комбинаций из заданного числа букв), тем самым совместив понятия естественности языка и энтропии.

$$H(x) = -\sum_i P_i \cdot \log_2 P_i, \quad (4)$$

где $H(x)$ – энтропия источника, характеризует избыточность языка;

P_i – частота появления i-го символа алфавита в тексте.

Максимальное значение энтропии достигается в случае, если частоты появления всех символов алфавита в любом тексте равны, т.е. сообщение по сути представляет собой набор случайных символов.

Для естественного языка энтропия всегда значительно меньше. Рассмотрим пример текста на русском языке:

ПРИВЕТ ПЕТР.

Т.к. естественный язык избыточен, то мы можем удалить часть символов, при этом текст все еще остается осмысленным:

ПР_ВЕТ П_ТР.

Фактически, это означает что энтропия H (x), посчитанная для подчеркнутых символов, близка к нулю (значение энтропии H (x)=0 означает, что мы можем с вероятностью 1 угадать символ алфавита, стоящий в тексте на месте подчеркивания).

Далее, Шеннон ввел следующую характеристику для произвольного шифра: ***расстояние единственности.***

Расстояние единственности показывает нам не то, какой размер должен иметь перехваченный шифротекст, чтобы его было легко дешифровать, а то,

насколько большим он должен быть, чтобы было возможно в принципе его однозначное дешифрование.

Для того чтобы затруднить противнику определение ключа и дешифрование сообщений, необходимо увеличивать (а желательно доводить до бесконечности) расстояние единственности в применяемых шифрах.

Расстояние единственности, в свою очередь, прямо пропорционально энтропии ключа и обратно пропорционально избыточности шифруемого сообщения на естественном языке. Увеличить расстояние единственности, таким образом, можно двумя способами.

Если энтропия ключа равна бесконечности, то расстояние единственности шифра будет тоже равно бесконечности. Энтропия ключа тем больше, чем длиннее ключ. Следовательно, расстояние единственности для шифра Вижинера с бесконечным ключом равно бесконечности.

Как уже отмечалось выше, использовать шифр с бесконечно большим ключом практически нецелесообразно.

Второй способ увеличения расстояния единственности состоит в уменьшении избыточности исходного текста. Если избыточность сообщения равна нулю, то ключ никогда не будет определен, а зашифрованное сообщение вскрыто, так как расстояние единственности будет равно бесконечности. К сожалению, на практике такая ситуация невозможна, так как любое осмысленное сообщение будет иметь некоторую отличную от нуля избыточность.

2.3 Критерий Шеннона надежности криптосистемы

Существует несколько (два) способов изложения критерия.

Идеальный шифр – это шифр, для которого выполняется условие: нарушитель, имея в своем распоряжении любое число шифротекстов, не может получить никакой дополнительной информации о неизвестных ему открытых текстах.

Фактически нарушитель с таким же успехом, что и при анализе перехваченных шифротекстов, может пытаться расшифровать сообщение, бросая монетку или перебирая все возможные ключи. Иными словами, криптосистема является практически надежной, если атака прямым перебором для нее вычислительно неэффективна и не существует других видов атак.

2.2.1 «Идеальный» шифр Виженера согласно требованиям Шеннону

Рассмотрим условия реализации шифра Вижинера, при которых он будет идеальным по определению Шеннона:

1) ключ шифрования по длине больше, чем шифруемое сообщение; для каждого нового сообщения ключ необходимо изменять, чтобы ключ не зацикливался;
2) все символы ключа генерируются случайно, не подчиняясь какому-либо правилу, независимо друг от друга;
3) частота появлений всех возможных символов ключа одинакова.

Пример: стационарный белый шум с равномерным распределением – ключ.

Шифр Виженера работает по принципу сложения по mod, т.е. (a+b)%33 – для русского алфавита. Данная операция обладает т.н. свойством *«отбеливания»*

(whitening) – если ключом является белый шум, то при шифровании любого сообщения с любыми статистическими свойствами результат шифрования также *неотличим (статистически) от белого шума*.

Идеального шифра Вижженера по Шеннону на практике не существует – т.к. для его работы требуется надежно хранить и передавать весь случайно сгенерированный ключ целиком. На практике в данном случае проще передавать по защищенному каналу само открытое сообщение – т.к. его объем равен объему ключа.

Для современных шифров применяется следующая схема – ключ шифрования имеет малый объем и передается единожды (для нескольких сообщений) по защищенному от перехвата каналу связи. Зашифрованное сообщение передается по открытому каналу, обычно имеющему большую пропускную способность и меньшее время задержки по сравнению с защищенным каналом.

Шифр является **практически стойким**, если на данный момент не существует известного алгоритма его взлома.

ЛЕКЦИЯ 3. КЛАССИФИКАЦИЯ АТАК НА ШИФРЫ

3.1 Классификация атак по возможностям атакующего

Атаки на криптографические системы можно разделить на две категории: атака с *активным атакующим*, атака *с пассивным атакующим*.

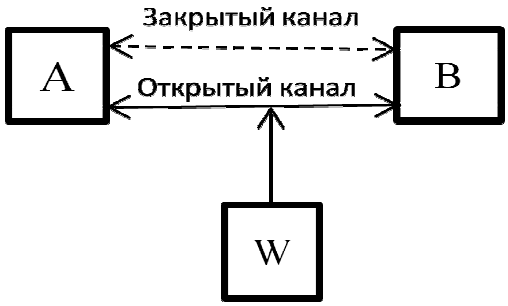


Рисунок 5. Схема криптографической защиты

Рассмотри типичную схему передачи зашифрованных данных (рис. 5). A и B – отправитель зашифрованного сообщения, соответственно. W – атакующий, целью которого могут являться: *ознакомление с содержанием передаваемых сообщений (дешифрование)*, *несанкционированное изменение сообщений* (такое изменение, при котором получатель не может обнаружить факт модификации и считает сообщение подлинным), *удаление сообщений и имитация сообщений* (передать сообщение через открытый канал таким образом, чтобы получатель считал сообщение подлинным сообщением от A)

Открытый канал связи – канал с большой пропускной способностью, высокой оперативностью передачи данных, однако он не защищен от вмешательства постороннего (*W*) – атакующего. Например, физические каналы интернет-доступа можно считать открытым каналом связи.

Закрытый канал связи – канал, гарантированно защищенный от вмешательства атакующего. При этом на практике чаще всего защищенный канал обладает более низкой, по сравнению с открытым каналом, скоростью передачи данных, большой задержкой.

A и *B* по закрытому каналу связи передают ключ, по открытому – зашифрованное сообщение. Атака с ***активным атакующим*** означает, что *W* полностью контролирует открытый канал связи, то есть может читать, вносить искажения, удалять или подменять эти данные, а также создавать сообщения, шифровать и передавать в канал связи. В то время как, ***пассивный атакующий*** может только считывать зашифрованные сообщения из открытого канала, никак не модифицируя и не задерживая/ не удаляя их. Ни активный, ни пассивный атакующий не могут получить доступ к закрытому каналу связи.

3.2 Классификация по информации, доступной атакующему

1. Атака с известным зашифрованным сообщением, т.е. у атакующего есть одно или несколько зашифрованных сообщений, но нет соответствующих расшифрованных сообщений.

2. Атака с известной парой открытое сообщение – зашифрованное сообщение, т.е. атакующему известно одно или несколько зашифрованных сообщений и соответствующие им открытые сообщения.
3. Атака с выбранным (навязанным) открытым сообщением, т.е. атакующий может выбрать любое открытое сообщение и затем перехватить результат его зашифрования.
4. Атака с выбранным (навязанным) зашифрованным текстом, т.е. атакующий может выбрать любое зашифрованное сообщение и затем перехватить результат его расшифрования.

Самой сложной для атакующего, но в тоже время наиболее «общей», т.е. применимой в большем числе практических случаев, является атака с известным зашифрованным сообщением. Если она реализована атакующим, т.е. если атакующий зная всего одно зашифрованное сообщение может дешифровать, модифицировать и имитировать любое другое сообщение (сформированное теми же участниками и с теми же параметрами шифра), то реализация всех остальных атак тривиальна.

Целью всех перечисленных атак чаще всего является вычисление многоразового ключа , используемого для шифрования. Знание ключа, таким образом, позволяет атакующему производить незаметно для отправителя и получателя расшифровку их сообщений, их модификацию и имитацию. Исключением может являться только атака с известным зашифрованным сообщением , целью которой может являться только расшифровка данного перехваченного сообщения, без вычисления ключа..

Атаки 1,2 – атаки с пассивным нарушителем.

Атаки 3,4 – атаки с активным нарушителем.

4. СОВРЕМЕННЫЕ СИММЕТРИЧНЫЕ ШИФРЫ

Главное отличие современных шифров, применяемых на практике, от идеального шифра Шеннона – это многоразовое использование ключа. При этом ключ должен иметь фиксировано малый размер, не зависящий от размеров передаваемого сообщения.

Шифр называется ***симметричным,*** если для операции зашифрования и расшифрования использовался один и тот же ключ. ***Ассимитричный*** шифр – шифр, который использует разные ключи для зашифрования и расшифрования.

Симметричные шифры делятся на поточные и блочные. ***Блочный шифр*** работает с блоком данных (сообщением) строго фиксированного размера (постоянного размера). Если длина сообщения не соответствует размеру блока, требуемого шифром, то применяется специальная процедура ***padding*** – т.е. процедура дополнения сообщения до нужной длины (обычно к сообщению дописываются нули).

Поточный шифр работает с потоком данных произвольной длины (чаще всего – с потоком бит). Например, поточный шифр может корректно зашифровать сообщения длиной 64, 128, 256, 259 бит, и ни для одного из них процедура ***padding*** не понадобится.

4.1 Симметричные блочные шифры

Одним из предложений Шеннона по повышению стойкости шифров было использование т.н. составных, или итеративных шифров, шифры. ***Составной шифр*** – шифр, который в ходе операций зашифрования и расшифрования производит многократные операции подстановки и перестановки над одним и тем же блоком данных (шифруемым сообщением).

Концепция составного шифра должна была дать возможность шифрам иметь два важных свойства: рассеяние (diffusion) и смешивания (confusion).

Пример: пусть необходимо зашифровать блок из 8 бит:

| 0 0 0 0 0 0 1 0 |
↓
|1 0 0 1 0 0 0 1 |

Изменение всего 1 бита открытого текста или ключа может привести к изменению любого бита зашифрованного текста – это рассеивание. Иными словами, ***рассеяние*** «запутывает» отношение между открытым и зашифрованным сообщениями. Шифр Вижженера под свойство рассеивания не попадает, это же можно сказать обо всех классических шифрах подстановки – для них всегда выполняется простое правило «одна буква зашифрованного – одна буква отрытого текста».

Свойство с***мешивания*** состоит в том, что оно должно скрыть отношения между зашифрованным текстом и ключом, или, другими словами, не дать атакующему, знающему пару открытое сообщение – зашифрованное сообщение, возможности вычислить правило (формулу) по которому происходило зашифрование.

Пример: шифр Виженера можно записать в виде простой формулы

$$(a + b) mod\ 32, \quad (6)$$

это означает, что имея всего одну пару открытое сообщение-зашифрованное сообщение, атакующий может вычислить это правило преобразования и, соответственно, определить ключ.

4.2 Итеративный блочный шифр. SP-сеть

Свойства рассеивания и смешивания – это свойства, которыми должен обладать шифр. Проверка наличия этих свойств осуществляется только на готовом шифре, т.к. заранее спроектировать шифр с такими свойствами труднодостижимо. Проверка указанных свойств, по большей части, проводится методом «прямого перебора» - на вход шифра подается большое количество открытых сообщений и ключей, соответствующие им зашифрованные сообщения анализируются и по итогам такого анализа вычисляются усредненные показатели рассеяния и смешивания. При этом, чем с большими блоками входных данных работает шифр – тем сложнее производить такой «переборный» анализ (например, у шифра ГОСТ-28147-89 возможно 2^{64} различных открытых сообщений для шифрования и 2^{256} различных ключей – проверить их все невозможно за обозримое время)

Чтобы обеспечить требуемые свойства современного блочного шифра, такие, как рассеяние и смешивание информации, этот шифр формируется как комбинация модулей транспозиции (P-блоки), модулей подстановки (S-блоки) и некоторыми другими блоками. При этом гарантированными свойствами рассеяния и смешивания обычно обладают только S-блоки, оперирующие над входными сообщениями малого размера.

Далее рассмотрим ***SP-сеть*** – сложно-составной шифр, чередующий операции перестановки и подстановки над блоком шифруемого текста.

P-блок – блок перестановки, подобен традиционному шифру транспозиции символов (перемещает биты).

S-блок – блок подстановки, можно представить себе как миниатюрный шифр подстановки. Этот блок может иметь различное число входов и выходов.

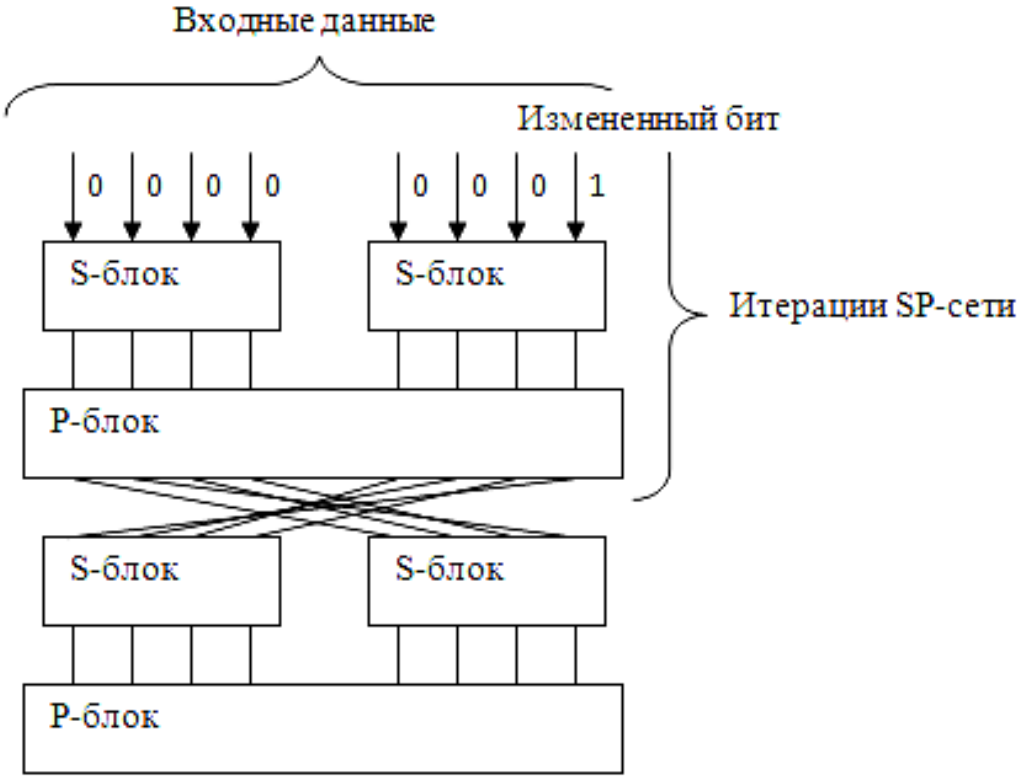


Рисунок 7. Итерации SP-сети

Каждая итерация – один «маленький» шифр.

Пример: рассмотрим 1-ую итерацию *S*, размер блока шифра равен 8 бит. На этапе подстановки 8 бит разбиваются на подблоки фиксированной длины, т.е.8 бит на 2 блока по 4 бита (см. рис. 7). После разбиения к данным внутри каждого блока применяется зависящая от ключа подстановка (S-блок). Существуют различные способы сделать S-блок и зависящим от ключа шифрования, и одновременно надежным с точки зрения рассеяния и смешивания:

1) генерирование списка «надежных S-блоков» - необходимо найти все возможные табличные подстановки для 4 битовых комбинаций. Таких таблиц будет порядка $(2^4)!$. Каждая таблица подстановки проверяется на соответствие свойствам рассеяния и смешивания, те подстановки, которые удовлетворяют заданным свойствам – нумеруем, и при шифровании выбираем таблицу подстановки с тем *номером*, который определяет ключ шифрования.
2) сделать зависимую от ключа подстановку, а именно:
   - сгенерировать проверить на рассеяние и смешивание один S-блок;
   - при шифровании с использованием ключа *K* мы шифруемую информацию *XOR*им с ключом (ключ должен быть того же размера, то и шифруемая информация, т.е. в указанном выше примере для каждого S-блока на каждой итерации нужно 3 бита ключа).

На практике более растространен 2 способ, как менее вычислительно сложный и не требующий дополнительной памяти для хранения всех используемых S-блоков.

После того, как S-блок выполнен, данные поступают на P-блок перестановки. В зависимости от ключа перестановки перемешиваем поступившие биты. Перестановка действует на все 8 бит.

4.2 Лавинный эффект

Лавинный эффект означает, что небольшие изменения в исходном тексте (или ключе) могут вызвать значительные изменения в зашифрованном тексте. Чем «больше» лавинный эффект, тем выше надёжность шифра и тем меньше итераций шифра требуется для обеспечения рассеяния и смешивания.
Пример: рассмотрим более подробно на примере изменения всего 1 бита во входных данных шифра (т.е. в отрытом сообщении). На первой итерации в результате выполнения S-блока ***1 изменившийся бит*** даст, в силу свойств рассеяния и смешивания отдельного S-блока, ***несколько*** изменившихся бит внутри своего блока. Изобразим графически:

Переходим на P-блок: 2 изменившихся бита одного блока разносятся по разным блокам, 1 изменившийся бит в каждом блоке дал по 2 изменившихся бита.

Для иллюстрации рассмотрим схему рассеивания и смешивания SP-сети с несколькими раундами. Как видно из схемы (рис. 9) изменение входных бит (изменившиеся биты отмечено жирными линиями между блоками) влечет за собой изменение нескольких бит на выходе, при этом ни количество изменившихся на выходе бит, ни их позиции в блоке нельзя предугадать заранее и вычислить по какому-либо простому правилу.

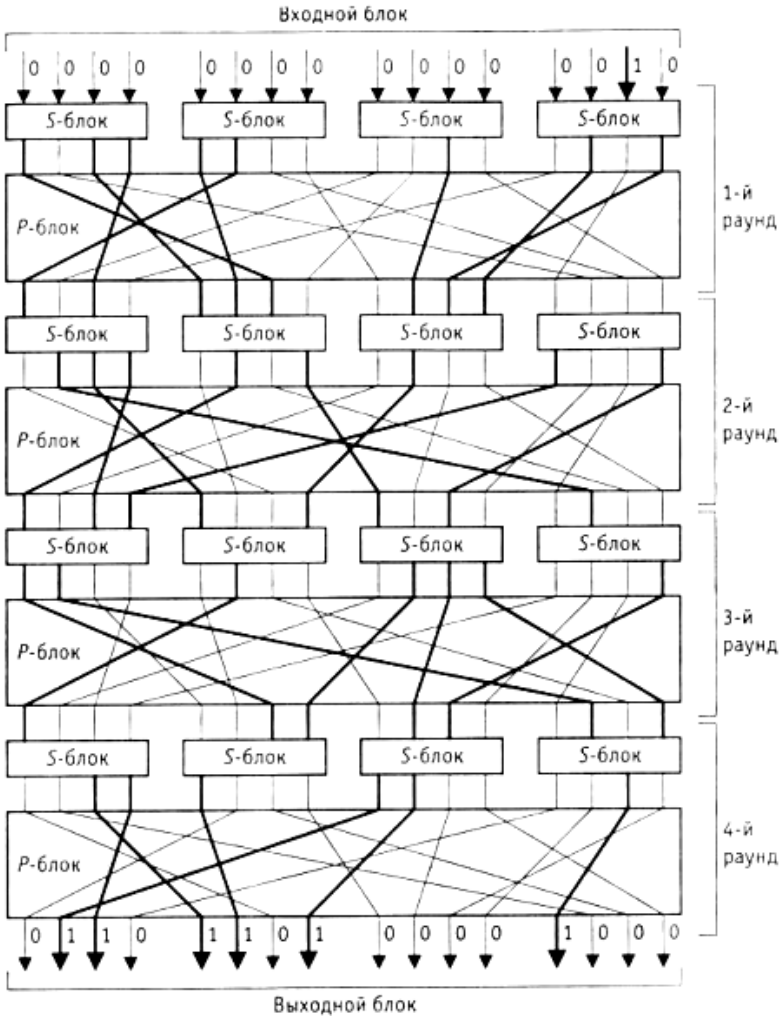


Рисунок 9. Рассеивание и смешивание в SP-сети, обусловленное лавинным эффектом

Другими словами, для итеративных блочных шифров ***лавинный эффект*** – это свойство, позволяющее за счет комбинирования этапа *S* и *P* достичь эффекта рассеивания для всего шифра в целом, при том, что каждый этап *S* обеспечивает эффект рассеивания только внутри малого блока. Чем больше размер одного блока подстановки и чем меньше количество таких блоков, тем быстрее лавинный эффект. Чтобы ускорить лавинный эффект, необходимо уменьшить число блоков на *S*-этапе. Если размер всего блока (общего) входящего текста фиксирован, то для ускорения лавинного эффекта придется наращивать размер малого блока подстановки.

Строгий лавинный эффект – это лавинный эффект, при котором изменение одного входного бита изменит с вероятностью ½ каждый выходной бит строгого лавинного эфекта.

4.4 Сеть Фейстеля

Сетью Фейстеля называется метод обратимых преобразований, при котором значение, вычисленное от одной из частей данных, накладывается на другие части. В большинстве случаев структура сети такова, что для шифрования и дешифрования используется один и тот же алгоритм – различие состоит только в порядке использования бит ключа.

На рисунке показана структура шифра, предложенного Х. Фейстелем. На вход алгоритма шифрования подается блок открытого текста длиной 64 бита и ключ *K*. Блок открытого текста делится на 2 равные части N_1 и N_2, которые последовательно проходят через 32 раунда обработки, а затем объединяются снова для получения блока шифрованного текста соответствующей длины.

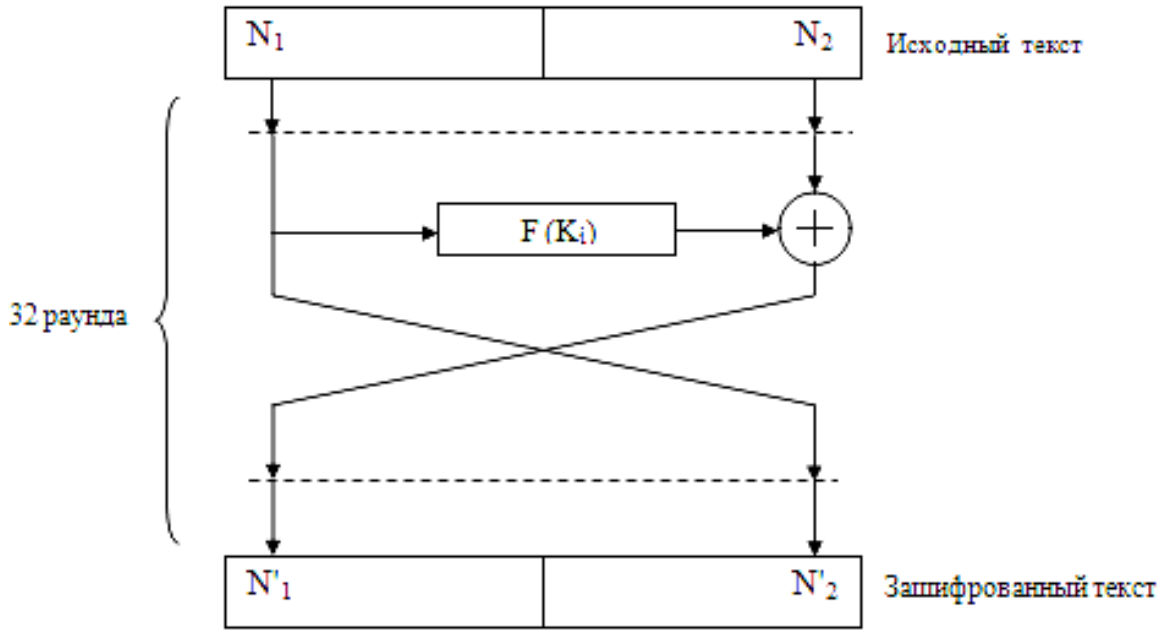


Рисунок 10. Классическая сеть Фейстеля

Все раунды обработки проходят по одной и той же схеме:

1) Левая половина входного блока, N_1, , без каких-либо изменений копируется в правую половину выходного блока N'_2;
2) После этого к левой половине блока данных применяется некоторая функция раунда *F*, *зависящая от ключа (*чаще всего функция раунда включает использование S-блоков*)* ;
3) Далее производится сложение полученного результата с правой половиной блока данных, например с помощью операции «побитовое исключающее или» (*XOR*). Результат сложения записывается в левую половину выходного блока, N'_1

Процесс дешифрования Фейстеля принципиально не отличается от процесса шифрования. Применяется тот же алгоритм, но на вход подается шифрованный текст, а подключи K_i используются в обратной последовательности.

Сеть Фейстеля имеет ряд преимуществ:

- малое количество настраиваемых параметров, что исключает возможность ошибки и выбора «ненадежных» параметров шифра;
- отсутствуют зависимые от ключа S-блоки;
- отсутствует сложное разбиение на подблоки.

4.5 Шифр ГОСТ 28147-89

Отечественный алгоритм шифрования ГОСТ 28147-89 определен в одноименном стандарте и используется более 20 лет. Алгоритм шифрует данные 64-битными блоками с использованием 256-битного ключа шифрования.

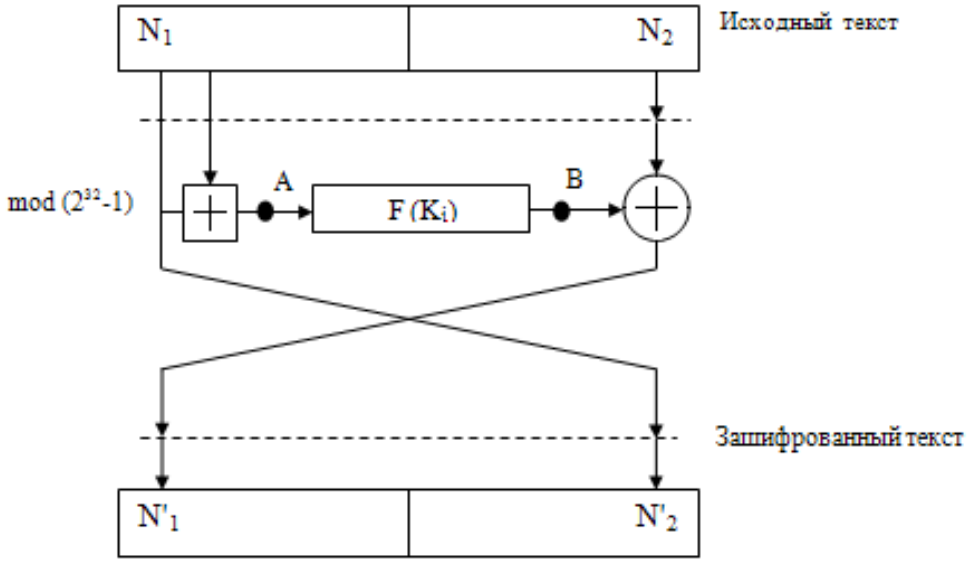


Рисунок 11. Схема раунда шифра ГОСТ 28147-89

Шифр реализует уже описанную ранее итеративную схему – сеть Фейстеля. При шифровании выполняется 32 раунда (итерации) преобразований, в каждом раунде предусмотрены следующие операции:
1) один из 32-битных подблоков данных складывается с 32-битным значением ключа раунда K_i по модулю 2^{32};
2) результат предыдущей операции разбивается на 8 фрагментов по 4 бита, которые параллельно «прогоняются» через 8 таблиц замен $S_0,..,S_7$;
3) 4-битные фрагменты (после таблиц замен) объединяются обратно в 32-битный подблок, битовое значение которого циклически сдвигается влево на 11 бит;
4) обработанный предыдущими операциями подблок накладывается на необработанный с помощью побитовой логической операции «исключающее или» (*XOR*);
5) подблоки меняются местами.

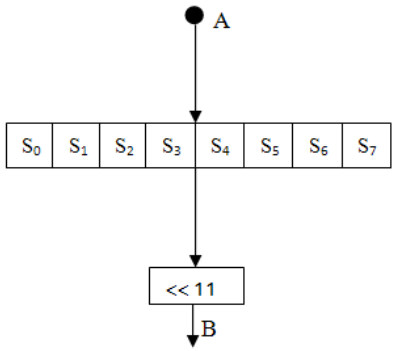


Рисунок 12. Разбиение блока на подблоки

Процедура расширения ключа в алгоритме ГОСТ 28147-89, фактически, отсутствует: в раундах шифрования последовательно используются 32-битные фрагменты K_0, …, K_7 исходного 256-битного ключа шифрования в следующем порядке: K_0, K_1, K_2, …, K_7 – за исключением последних 8 раундов – в раундах с 25-го по 31-й фрагменты используются в обратном порядке.

Расшифрование полностью аналогично зашифрованию, но с другим порядком использования фрагментов ключа:

- в прямом порядке – в первых 8 раундах;
- в остальных раундах – в обратном порядке.

4.6 Режимы работы блочного шифра ГОСТ 28147-89

ГОСТ 28147-89 предусматривает следующие режимы шифрования данных:

1) простая замена;
2) сцепление блоков;
2) гаммирование;
3) гаммирование с обратной связью.

В любом из этих режимов данные обрабатываются блоками по 64 бита, на которые разбивается массив, подвергаемый криптографическому преобразованию (собственно, именно поэтому ГОСТ 28147-89 относится к блочным шифрам).

4.6.1 Режим простой замены (Electronic Code Book)

Шифруемые данные разбиваются на блоки по 64 бита. Каждый блок исходного текста шифруется независимо от других. Стойкость режима простой замены равна стойкости самого шифра. Однако, структура исходного текста при этом не скрывается. Зашифрование двух совпадающих блоков открытого текста приводит к появлению двух одинакового блока зашифрованного текста. Атакующий может легко манипулировать зашифрованным сообщением путем удаления, повторения или перестановки блоков; зашифрованное сообщение после таких манипуляций будет корректно расшифровано и факт манипуляций атакующего не будет выявлен. Схемы

алгоритмов зашифрования и расшифрования в режиме простой замены приведены на рисунке 13.

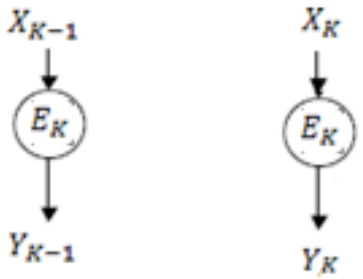


Рисунок 13. Режим простой замены

$$D_K(Y_K) = X_K;\ \ E_K(X_K) = Y_K. \quad (7)$$

Режим простой замены не рекомендуется использовать для шифрования больших блоков данных, т.к. у него присутствует ряд недостатков:

1) возможность статистической атаки на избыточные данные большого объема;
2) не обеспечивает защиту от подмены блоков;
3) если длина шифруемого массива данных не кратна 8 байтам или 64 битам, возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит.

4.6.2 Режим сцепления блоков (Ciher Block Chaining)

В режиме сцепления блоков шифрованного текста (CBC) каждый блок исходного текста складывается побитово по модулю 2 с предыдущим блоком **зашифрованного текста**, и только затем шифруется. Для начала процесса шифрования используется ***синхропосылка*** (или ***вектор инициализации***), которая передается в канал связи в открытом виде. Вектор инициализации в этой схеме используется в качестве предыдущего блока для шифрования самого первого блока сообщения.

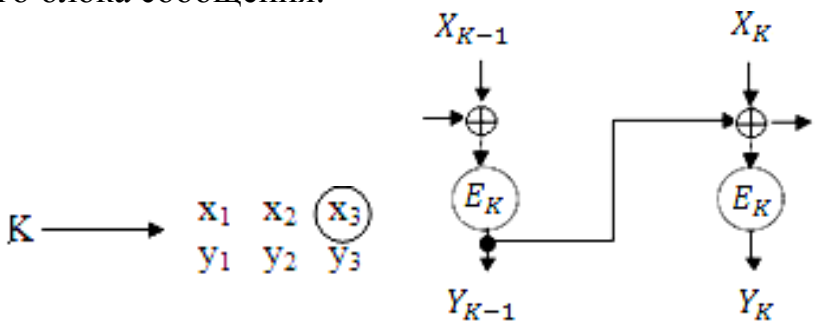


Рисунок 14. Режим сцепления блоков шифрованного текста

где x_3 перед подачей на шифрование складывается по модулю 2 с предыдущим уже зашифрованным текстом.

$$Y_K = E_K(Y_{K-1} \oplus X_K), \quad (8)$$

где E_K – операция шифрования блока с ключом *K*.

$$D_K(Y_K) \oplus Y_{K-1} = X_K. \quad (9)$$

При использовании данного режима структура и избыточность исходного текста скрывается за счет сложения предыдущего блока шифрованного текста с очередным блоком открытого текста. Таким образом, например, даже если шифруется длинная последовательность нулей, результат шифрования не будет содержать повторяющихся блоков (как в случае с режимом простой замены).

Основные преимущества:

1) фиксированный блок из всех будет шифроваться по-разному, нельзя вывести однозначное соответствие между незашифрованным и зашифрованным текстом;

4.6.3 Режим гаммирования (Output Feedback)

Режим гаммирования подобен режиму сцепления блоков за исключением того, что величины, складываемые по модулю 2 с блоками исходного текста, генерируются независимо от исходного или шифрованного текста.

Зашифрованный: $Y_i = X_i \oplus K_i$, (10)

Расшифрованный: $X_i = Y_i \oplus K_i$, (11)

где K_i – т.н. «гамма» (keystream) – поток псевдослучайных бит, сгенерированных шифром на основе заданного ключа..

Для начала процесса шифрования также используется вектор инициализации (IV)

$$E_K(IV) \rightarrow K_1, E_K(K_1) \rightarrow K_2,$$

)

Суть гаммирования состоит в том, чтобы не шифровать сами данные с помошью блочного шифра, а использовать блочный шифр для генерации псевдослучайной последовательности – гаммы, и затем «накладывать» получившуюся гамму на массив данных с помощью операции исключающего ИЛИ (*XOR*). При этом операция зашифрования ничем не отличается от операции расшифрования, т.к. наложение той же гаммы на массив данных с помощью XOR два раза подряд дает исходный массив данных без искажений.

Особенностью данного режима является его схожесть с т.н. поточными шифрами. В частности, все вычислительно сложные операции шифра выполняются при генерации гаммы, а непосредственно шифрование производится с помощью простейшей операции XOR. В данном случае на практике возможно сгенерировать гамму «впрок», т.е. выполнить все вычислительно сложные операции, связанные с шифрованием данных, еще до того, как эти данные стали известны.

Режим гаммирования обладает рядом преимуществ:

1) любые битовые ошибки, возникшие в процессе передачи, не влияют на расшифрование последующих блоков; при ошибке передачи в N битах

зашифрованного сообщения – ошибки расшифровки будут тоже только в N битах (для ранее рассмотренных режимов данные повреждаются блоками, т.е. при повреждении всегододного блока неправильно расшифруется весь блок)
2) алгоритм зашифрования и расшифрования очень прост - XOR;
3) шифрование возможно для любого размера входной последовательности;

4.6.4 Режим гаммирования с обратной связью

В случае современных блочных шифров применение режима гаммирования может привести к ослаблению шифров – т.к. любой объем псевдослучайной гаммы, по сути, генерируется на основе всего 64 бит вектора инициализации (возможно, что путем стат. анализа гаммы удастся восстановить исходный вектор инициализации?). Альтернативой режиму гаммирования является гаммирование с обратной связью.

Гаммирование с обратной связью позволяет, как и режим сцепления блоков, сделать шифрование каждого блока зависимым от результатов шифрования предыдущих, т.е. позволяет для генерации гаммы использовать не только исходный вектор инициализации, но и большой массив зашифрованных блоков. Гаммирование с обратной связью отличается от простого гаммирования тем, что очередной блок гаммы K_i вырабатывается как результат преобразования зашифрования предыдущего блока зашифрованных данных, а для зашифрования первого блока массива данных элемент гаммы вырабатывается как результат преобразования вектора инициализации (или, в терминах ГОСТ-281470-89, синхропосылки).
Преимущества:
- в перспективе – сложнее обнаружить закономерность в гамме.
Недостатки:
- отсутствует возможность заранее генерировать гамму.

4.7 Пример схемы шифрования больших объемов данных

Tree Crypt, Best Crypt, Free OTFE – программы создания зашифрованных виртуальных дисков, на их примере можно рассмотреть как шифруются большие объемы данных..
Есть объемный блок данных, есть пароль, на основе пароля создается ключ, который шифрует этот большой объем данных.
Типовые задачи шифрования больших объемов данных:

1. Быстро уничтожить данные (для 1Тб ~ 40 ч. для надежного «затирания» данных на жестком диске);
2. Быстро сменить пароль к данным;
3. Множественность паролей (владелец одного пароля не знает других паролей).

В схему шифрования ввели отдельный служебный блок данных блок: ***Critical Data Block.***
Шифрование данных в этом случае производится следующим образом. Генерируется случайный ключ шифрования, которым шифруется весь массив пользовательских данных. Затем этот случайный ключ шифруется паролем

пользователя и записывается в блок ***Critical Data Block*** который хранится вместе с зашифрованными данными (рис. 15).

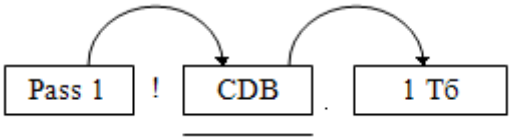


Рисунок 15. Схема шифрования больших объемов данных

Если нужно организовать доступ к данным, используя несколько независмых паролей – нужно просто создать соответсвующее множество блоков Critical Data Block, каждый из которых будет содержать один и тот же случайный ключ, зашифрованный разными паролями пользователя. Для быстрого ничтожения данных необходимо удалить олько Critical Data Block – даже при знании пароля пользователя нельзя будет расшифровать данные. Аналогично, для смены пароля достаточно перезаписать Critical Data Block.

ЛЕКЦИЯ 5. ПОТОЧНЫЕ ШИФРЫ

5.1 Общие сведения о поточных шифрах

Область применения

Поточные шифры представляют собой разновидность гаммирования и преобразуют открытый текст в шифрованный последовательно по 1 биту. При этом любой объем информации может быть зашифрован без дополнительных процедур дополнения нулями до заданного размера. Рассмотрим пример: возьмем 17 бит и зашифруем его по шифру ГОСТ 28147-89, работающему с блоком информации по 64 бита. В этом случаи необходимо воспользоваться операцией ***padding*** и дополнить 17 бит (17 бит + 47 нулей), так как 17 бит < 64 бит (размер блока шифра ГОСТ 28147-89). Надо понимать, что в этом случае избыточность передаваемых данных возрастет в 4 раза. Однако, при использовании поточного шифра такой сложности не возникнет, будет зашифровано ровно то количество бит, которое дано.

Генератор ключевого потока (keystream), иногда называемый ***генератором бегущего ключа***, выдает последовательность бит K_1, K_2, .., K_i,…Эта ключевая последовательность складывается по модулю 2 с последовательностью бит исходного текста X_1, X_2, ..., X_i,…для получения шифрованного текста:

$$Yi = Xi \oplus Ki. \quad (12)$$

Восстановление исходной последовательности заключается в повторном сложении по модулю 2 исходной гаммы (т.е. гаммы, заново сгенерированной получателем сообщения на основе секретного ключа) и полученного зашифрованного сообщения.

$$Xi = Yi \oplus Ki. \quad (13)$$

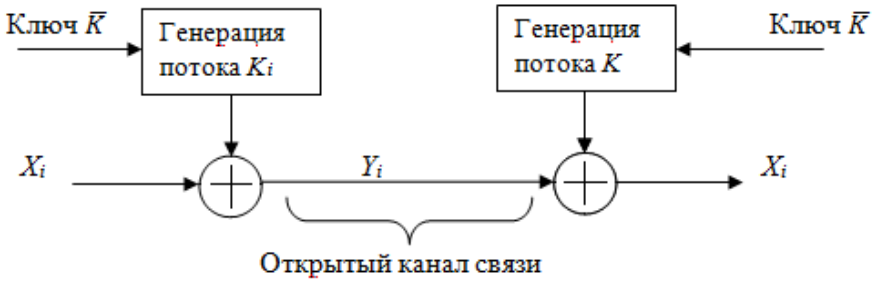


Рисунок 16. Общая схема поточного шифра

Ключ $\overline{K}$ – пользовательский пароль. Его длина не зависит от длины шифруемого текста. Потоковый шифр, показанный на рисунке 16, называется ***синхронным*** – шифр, в котором поток ключей генерируется независимо от открытого текста и шифртекста.

При шифровании генератор потока ключей выдаёт биты потока ключей, которые идентичны битам потока ключей при дешифровании. Потеря («выпадание») знака шифртекста приведёт к нарушению синхронизации между этими двумя генераторами и невозможности расшифрования

оставшейся части сообщения. Т.е., в данном случае получатель будет считать, что ему пришел *999й* бит зашифрованного сообщения и будет пытаться его расшифровать *999м* битом гаммы, хотя на самом деле принятый бит был *1000м* в сообщении и ему соответсвенно должен быть поставлен в соответсвие *1000й* бит гаммы. Очевидно, что в этой ситуации отправитель и получатель должны повторно синхронизироваться (согласовать номера передаваемых бит) для продолжения работы. Обычно синхронизация производится вставкой в передаваемое сообщение специальных служебных маркеров (счетчиков бит). В результате этого пропущенный при передаче знак приводит к неверному расшифрованию лишь до тех пор, пока не будет принят один из маркеров.
Плюсы синхронного поточного шифра:
- отсутствие эффекта распространения ошибок (только искажённый бит будет расшифрован неверно);
- предохраняют от любых вставок и удалений шифротекста, так как они приведут к потере синхронизации и будут обнаружены.
Минусы синхронного поточного шифра:
- уязвим к изменению отдельных бит шифрованного текста. Если злоумышленнику известно открытое сообщение или его часть, он может изменить известные ему биты в зашифрованном сообщении так, чтобы они расшифровывались, как ему нужно (т.е. фактически если атакующему нужно в передаваемом сообщении содержательно изменить несколько бит – он просто меняет эти биты в зашифрованном сообщении).

Особенности:

1) Если несколько бит выпало при передаче через открытый канал, то шифр не работает;

5.2 Генератор ключевого потока

Структуру генератора ключевого потока можно представить в виде конечного автомата с памятью, состоящего из трех блоков:

(1) - блока памяти, хранящего информацию о внутреннем состоянии генератора;
(2) - выходной функции, генерирующей бит ключевой последовательности в зависимости от состояния;
(3) - функции следующего состояния (функции переходов), задающей новое состояние, в которое перейдет генератор на следующем шаге.

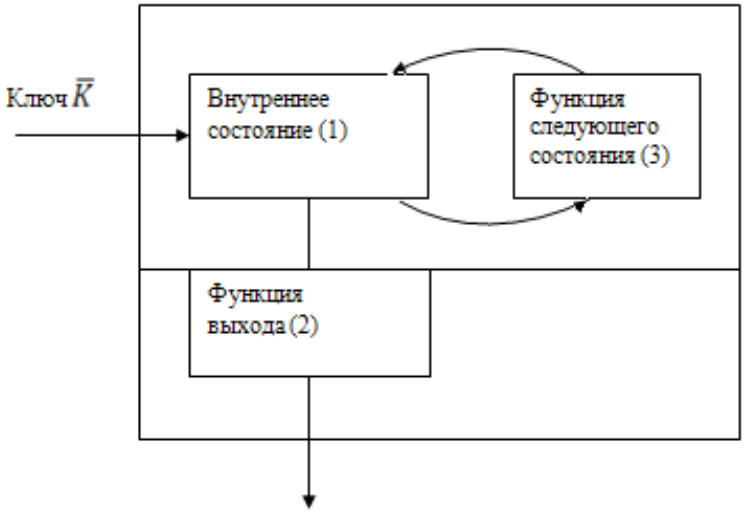


Рисунок 17. Структура генератора ключевого потока

Рассмотрим на примере: пусть внутреннее состояние генератора имеет вид

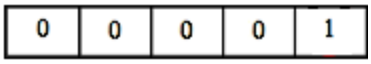


Рисунок 18. Пример внутреннего состояния генератора

Функция выхода - сжимает до 1 бита – например, ***бит четности***; если число едениц во внутреннем состоянии четное, то →0, если нечетное, то →1. Функция следующего состояния из этих 5 бит должна получить 5 бит следующего состояния (например, сдвинуть влево на 1 позицию, а на правую позицию записать бит четности)

Рисунок 19. Внутреннее состояние генератора после обновления

Абсолютно все генераторы ключевого потока зацикливаются (начинают повторяться через некоторый интервал/период *T*). Любой генератор потока, если его на разных шагах запустить с одинаковым внутренним состоянием - выдаст одно и то же. Очередной выходной бит однозначно определяется внутренним состоянием и функцией выхода, поэтому максимальный период генератора ключей определяется числом возможных внутренних состояний.

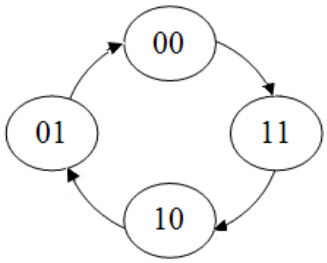


Рисунок 19. Последовательность переключения внутренних состояний генератора

Если *n* количество ячеек памяти, хранящих внутреннее состояние, то максимальный период $T=2^n$.

5.3 Самосинхронизирующийся поточный шифр

Идея ***самосинхронизирующихся поточных шифров*** (или ***шифрования с автоключом*** – CipherText Auto Key (CTAK)) заключается в

том, что внутреннее состояние генератора является функцией фиксированного числа предшествующих битов шифрованного текста. Поскольку внутреннее состояние зависит только от *n* бит шифрованного текста, генератор на принимающей стороне войдет в синхронизм с передающей стороной после получения *n* бит.

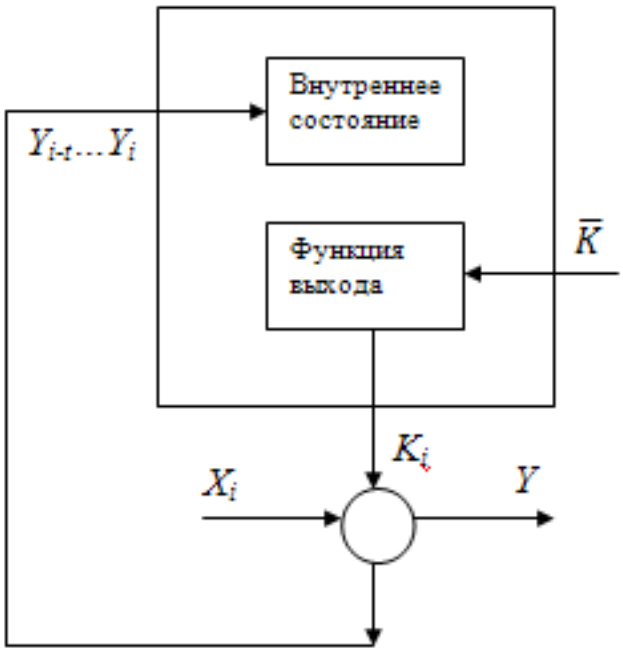


Рисунок 20.

Реализация этого подхода выглядит следующем образом: чтобы сгенерировать очередной бит ключевого потока, необходимо записать во внутреннее состоянии генератора *N* предыдущих бит уже зашифрованного текста. Самосинхронизирующийся шифр позволяет восстанавливать после ошибок синхронизации (выпадание бита из потока) – любой «выпавший» бит приведет к неправильной расшифровке соответсвующего бита открытого сообщения и *N* следующих за ним бит .

Недостатком системы является распростронение ошибок – изменение или «выпадание» одного бита зашифрованного сообщения приведет к появлению *N+1* неправильно расшифрованных бит.

ЛЕКЦИЯ 6. СХЕМЫ ГЕНЕРАТОРОВ КЛЮЧЕВОГО ПОТОКА

6.1 Линейный конгруэнтный генератор

Генераторы псевдослучайных чисел могут работать по разным алгоритмам. Одним из простейших генераторов является так называемый линейный ***конгруэнтный генератор***, который для вычисления очередного числа X_i использует формулу

$$X_i = (aX_{i-1} + b) mod\ N, \qquad (15)$$

формула (14) отражает функцию изменения внутреннего состояния, где mod – остаток от деления, N- модуль, a – множитель, b – приращение, X_{i-1} – предыдущее псевдослучайное число, являются неотрицательными целыми числами.

Существуют две упрощенные модификации:

1) Если b =0, то линейный конгруентный генератор называется ***мультипликативным;***

$$X_i = (aX_{i-1}) mod\ N, \qquad (16)$$

2) Если $a=1$, то ***аддитивным***.

$$X_i = (X_{i-1} + b) mod\ N. \qquad (17)$$

состояние генератора X_i – число от 0 до $N-1$ включительно. $X_i = 0, N-1$, $X_i \in [0, N-1]$. Ключ $K=X_0$.

Рассмотрим на примере:

$$X_i = (X_{i-1} + 7)^5 mod\ 11, \qquad (18)$$

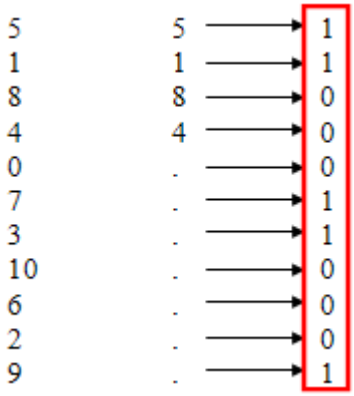


Рисунок 21. Пример генерации гаммы на основе конгруэнтного генератора

Где выделенные элементы – это сгенерированный бит гаммы (нечетное внутр. состояние дает бит гаммы 1, четное – 0).

Максимальная длина периода для аддитивного конгруентного генератора – N.

Для того, чтобы период был максимальным необходимо, чтобы b и N были взаимно простыми.

Для мультипликативного конгруентного генератора период равен ($N-1$). В мультипликативном конгруентном генераторе есть 1 нежелательное (плохое) внутреннее состояние – 0, если в него зашел, то последующие числа будут 0.

Наибольший период при взаимно простых a и N.

Преимуществом линейных конгруэнтных генераторов является их быстрота за счет малого количества операций на бит. К несчастью линейные конгруэнтные генераторы нельзя использовать в криптографии, так как они предсказуемы – атакующий, зная малое число бит гаммы, может предсказать все последующие. За малое число выходных данных генератора можно определить его внутреннее состояние, а значит, взломать шифр.

6.2 ***РСЛОС (Linear feedback shift register (LFSR))***

Регистр сдвига с обратной связью состоит из двух частей: регистра сдвига и функции обратной связи. Простейшим видом регистра сдвига с обратной связью является ***регистр сдвига с линейной обратной связью (РСЛОС).*** Обратная связь представляет собой XOR некоторых битов регистра; эти биты называются ***отводной последовательностью***.

Регистр сдвига с линейной обратной связью. В основе лежит рекуррентная последовательность вида

$$X_n = \sum_{i=1}^{K} a_i X_{n-i} + c, \quad (19)$$

где $a_i = \{0,1\}$, $X = \{0,1\}$.

Регистр сдвига представляет собой последовательность битов. Количество битов определяется длиной сдвигового регистра.

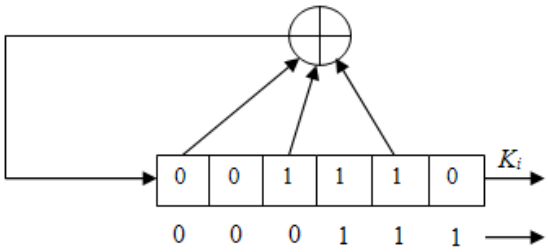


Рисунок 22. 6-ти битовый РСЛОС

Всякий раз, когда нужно извлечь бит, все биты регистра сдвига сдвигаются вправо на 1 позицию. Новый крайний левый бит получается применением функции XOR к битам обозначенным отводами. Крайний правый – выходное состояние. На выходе регистра оказывается один, обычно младший значащий бит.

Периодом регистра называется длина получаемой последовательности до начала ее повторения. РСЛОС (*n*-битовый) может находиться в одном из 2^n-1 внутренних состояний. Это означает, что теоретически такой регистр может генерировать псевдослучайную последовательность с периодом 2^n-1 битов. Только при определенных отводных последовательностях РСЛОС циклически пройдет через все 2^n-1 внутренних состояний. Последовательность выдаваемая генератором с максимальным периодом, называется ***М-последоваиельностью***. РСЛОС-гененратор в «идеальном» случае проходит все возможные внутренние состояния, кроме одного – все 0. Если любой РСЛОС-генератор инициализировать всеми нулями, то он не сможет выйти из этого состояния, так как XOR от 0 всегда 0.

От выбора выхода на XOR зависит период генератора.

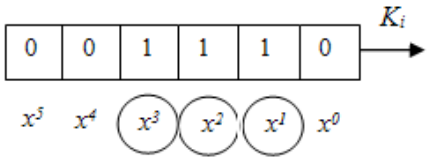


Рисунок 23. Нумерация ячеек РСЛОС

Весь генератор удобно описывать многочленом, описывающим с отводную последовательность бит для XOR.

Например, многочлен ($x^6 + x^5 + x^3 + x^1 + 1$) позволяет однозначно построить РСЛОС и определить структуру отводной последовательности.

Степень многочлена задает длину РСЛОС – для указанного выше многочлена РСЛОС будет состоять из 6ти бинарных ячеек памяти. Степени *x* многочлена, включая нулевую, задают отводную последовательность, отсчитываемую от правого края сдвигового регистра (нумерацию ячеек степенями *x* см. на рисунке). Для того, чтобы РСЛОС генератор давал максимальный период, еобходимо, чтобы характеристический многочлен был примитивен.

6.3 Комбинированные генераторы

При построении поточных шифров в качестве генератора потока чаще применяются различные комбинации нескольких регистров сдвига с линейными обратными связями. Наиболее часто встречаются узлы, называемые ***комбинирующими генераторами*** и (нелинейными) ***фильтр-генераторами***.

У комбинирующих генераторов в каждом такте работы очередные элементы выходных последовательностей нескольких регистров сдвига поступают на вход некоторой функции. Значение этой функции является выходом генератора (элементом гаммы).

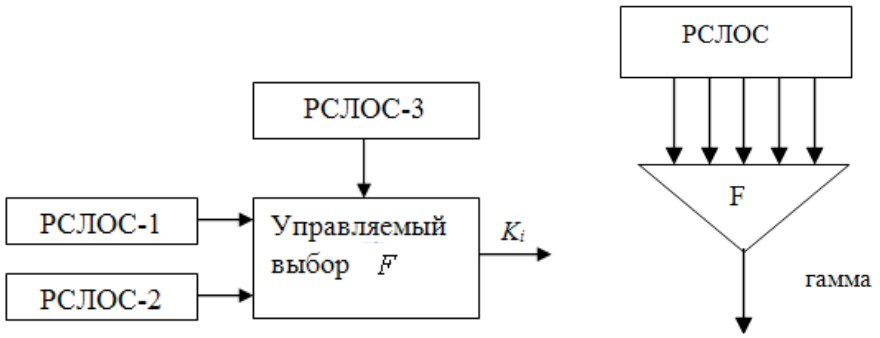


Рисунок 25. Комбинирующий генератор (слева). Фильтрующий генератор (справа)

В свою очередь, при объединении в криптосхему целые узлы, либо их части могут влиять друг на друга, изменяя заполнения некоторых регистров, а также управляя их движением. Обычно отдельные регистры сдвига в комбинированном генераторе работают синхронно, изменяя свое внутреннее состояние один раз в течение такта работы комбинированного генератора. Если же запуск\простой РСЛОС в ходе каждого такта работы комбинированного генератора зависит от состояния других РСЛОС или

вычисляется на основе какой-либо функции - такое движение называется управляемым или неравномерным. Подобная схема с управляемым движением, как правило, существенно «усложняет» выходную последовательность, т.е. атакующему сложнее предсказать следующий бит, зная несколько предыдущих.

6.4Схемы комбинированных генераторов на основе РСЛОС

6.4.1 Поточный шифр на базе РСЛОС (LFSR)

Основной подход при проектировании генератора потока ключей на базе LFSR прост. Сначала берется один или несколько LFSR, обычно с различными длинами и различными многочленами обратной связи. Если длины взаимно просты, а все многочлены обратной связи примитивны, то у образованного генератора будет максимальный период, равный произведению периодов РСЛОС в его составе. Ключ шифрования выступает в роли начального состояния регистров LFSR. Каждый раз, когда необходим новый бит, необходимо сдвинуть на бит регистры LFSR (это иногда называют ***тактированием*** (clocking)). Бит выхода представляет собой функцию, желательно нелинейную, некоторых битов регистров LFSR.

6.4.2 Поточные шифры на основе комбинирования нескольких РСЛОС. Прореживающий генератор

Рисунок 26. Схема прореживающего генератора

Количество регистров РСЛОС-1 и РСЛОС-2 обычно различаются, но у обоих генераторов в схеме общее тактирование (на один такт схемы оба генератора одновременно выдают по 1 биту на выходе). Если на выходе РСЛОС-2 сгенерирована 1, то сгенерированный РСЛОС-1 бит записывается в выходную гамму. Если на РСЛОС-2 – 0, то в гамму биты не записываются, запускается следующий такт.

Недостаток данной системы в том, что заранее невозможно определить, сколько тактов понадобится для того, чтобы сгенерировать заданное кол-во бит. Вариантом этого генератора является **прореживающий из трех РСЛОС**:

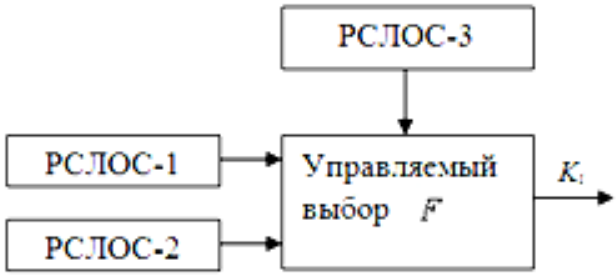


Рисунок 27. Схема прореживающего генератора из трех РСЛОС

У каждого генератора разное число ячеек памяти, разная структура обратной связи, но общее тактирование. Предположим, что РСЛОС-1 выбран селектирующим генератором (соответственно, ни один сгенерированный им бит не выдается в гамму). Если в ходе общего такта на выходе РСЛОС-1 появился 0, то в гамму выходит выходной бит РСЛОС-2, сгненерированный на данном такте. Иначе, такте если РСЛОС-1 выдал 1, то в гамму выходит бит от РСЛОС-3.

Генератор потока на каждом шаге дает по одному биту в гамму. F – функция выхода: из трех бит получает 1, идущий в гамму.

1.1 Самопрореживающий генератор РСЛОС

В основе схемы – один РСЛОС, который отрабатывает по два такта на 1 бит выходной гаммы. В зависимости от сгенерированных бит, на выход генератора (в гамму) бит выдается по следующему правилу:

$$\begin{cases} 01 \rightarrow \sqcup \\ 00 \rightarrow \sqcup \\ 10 \rightarrow 0 \\ 11 \rightarrow 1 \end{cases}$$

$$\text{РСЛОС} - 1 \rightarrow \textcircled{F} \rightarrow K_i$$

где РСЛОС-1 работает по два такта, после чего выдает два бита. Функция F дает один бит по следующему принципу:

- если из двух бит первый равен 0, то оба бита сбрасываются, и в гамму ничего не идет;
- если первый бит равен 1, то второй бит идет на выход.

6.4.3 Поточные шифры на основе схемы «Stop-and-Go» (с переменным тактированием)

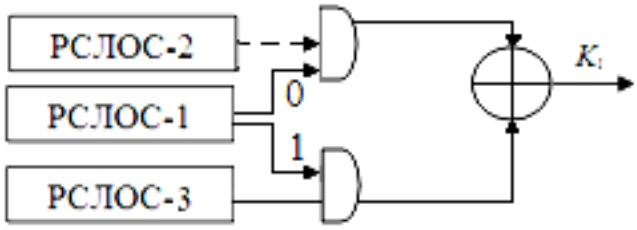


Рисунок 28. Схема поточного шифра на основе «Stop-and-Go»

Тактирование общее, но в на каждом шаге РСЛОС-2 и РСЛОС-3 могут пропускать такт, т.е. не изменять свое состояние. Есть РСЛОС-1, который управляет тактированием. Если РСЛОС-1 дал на выход 0, то отрабатывает такт только РСЛОС-2. В качестве гаммы при этом выдается XOR бита РСЛОС-2 на текущем такте и бита РСЛОС-3 на предыдущем такте. При этом РСЛОС-3 такт не отрабатывает, а его состояние остается прежним. Если РСЛОС-1 выдает 1 - все действия те же, но РСЛОС-2 и РСЛОС-3 меняются местами, то есть РСЛОС-3 – отрабатывает, а РСЛОС-2 – простаивает.

РСЛОС-1 работает на всех тактах. Этот генератор на данный момент не взломан, хотя найдены его слабые места, позволяющие частично восстановить внутреннее состояние и многочлены обратной связи.

6.4.4 Потоковые шифры на основе мажоритарного комбинирования РСЛОС

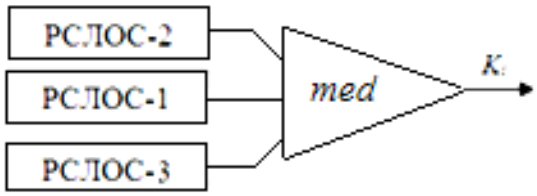


Рисунок 29. Схема поточного шифра на основе мажоритарного комбинирования

Для построения схемы берется произвольное нечетное кол-во разных РСЛОС с общим тактированием. На каждом такте работы собираются биты со всех РСЛОС, в гамму выдается значение, которое выдано большенством генераторов (метод голосования большенства).

6.4.5 Поточный шифр на основе мажоритарного голосования с переменным тактированием

РСЛОС, которые в меньшинстве при голосовании сгенерированных бит, пропускают следующий такт генерации (при этом простаивающие генераторы выдают бит на следующем такте в голосовании- просто этот бит равен биту на

предыдущем такте; внетренее состояние простаивающих генераторов не изменяется при этом). Схема аналогична рисунку 29.

6.3 Шифр RC4

RC4 - это потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях в тех случаях, когда удобнее шифровать поток не на уровне отдельных бит, а на уровне байт. Шифр RC4 применяется в некоторых широко распространённых стандартах и протоколах шифрования таких, как WEP, WPA и TLS.

Основа алгоритма шифра состоит из генератора гаммы, который выдаёт ключевой поток. За каждый такт потоковый шифр RC4 выдает 1 байт информации. Внутренее состояние шифра – 255 ячеек памяти, каждая ячейка содержит 1 байт

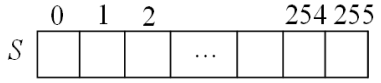


Рисунок 30 . Байтовые ячейки RC4

RC4 — фактически класс алгоритмов, определяемых размером ячейки (т.е. данный шифр можно реализовать и с байтовыми ячейками памяти, и с 7-битными, и с 6-битными и т.д.;). Параметр n - число бит в ячейке, является размером слова для алгоритма и определяет длину массива S (внутреннее состояние) . Если в шифре используются n-битные ячейки памяти, то число их в массиве S составляет 2^n.

Алгоритм состоит из двух этапов:

1. Инициализация массива S

Внутреннее состояние RC4 представляется в виде массива размером 2^n и двух счётчиков. Массив далее будет обозначаться как S. Он всегда содержит перестановку всех 2^n возможных значений n-битного слова. Также для работы шифра требуются два целочисленных счётчика (обозначены далее как i и j).

1.1Начальное заполнение массива:

$\boldsymbol{for\ i = 0\ to\ 2^n - 1}$,

$\boldsymbol{S[i] = i}$

1.2 Инициализация ключом (перестановка):

$j = 0$

$for\ i = 0\ to\ 2^n - 1$

$j = (j + S[i] + Key[i\ mod L] mod 2^n$,

$Swap(i, j)$## поменять значения i, j местами.

Key - массив, содержащий ключ шифрования.

1.3 Генерация псевдо-случайного слова K *(гаммы)*

Генератор ключевого потока RC4 на каждом шаге, во первых меняет местами два значения, хранящиеся в S (фаза обновления внутр. состояния) и затем выбирает значение из S в качестве гаммы. На одном такте RC4 генерируется одно n-битное слово K для гаммы, которое в последующем суммируется по модулю 2^n с исходным текстом (или, как вариант, просто XORится) для получения зашифрованного текста.

1.4 Генерация гаммы:

M – число байт, который должен получить генератор.

$i = 0, j = 0;$

$for\ m = 1\ to\ M;$

$\gamma = (i + 1) mod 2^n;$
$j = (j + S[i]) mod 2^n,;$
$Swap(i, j);$
$K[m] = S[(S[i] + S[j]) mod 2^n];$
end

ЛЕКЦИЯ 7. ШИФРОВАНИЕ С ОТКРЫТЫМ КЛЮЧОМ

Алгоритмы ***шифрования с открытым ключом***, также называемые ***асимметричными алгоритмами шифрования***, устроены так, что ключ, используемый для зашифрования сообщений, отличается от ключа, применяемого для их расшифрования. Более того, ключ расшифрования K_1 не может быть за обозримое время вычислен, исходя из ключа шифрования K_2. Свое название алгоритмы с открытым ключом получили благодаря тому, что ключ шифрования не требуется держать в тайне. Любой может им воспользоваться, чтобы зашифровать свое сообщение, но только обладатель соответствующего секретного ключа расшифрования будет в состоянии прочесть это шифрованное сообщение. Ключ шифрования обычно называют ***открытым ключом***, а ключ расшифрования — ***закрытым ключом***.

$$D_{K_1}\left(E_{K_2}(X)\right) = X,$$

Зашифровать сообщение может каждый участник, обладающй ключом K_2, а расшифровать может только участник с ключом K_1. Расшифровать исходные данные ключом K_2 невозможно.

7.1 Односторонние функции

Все алгоритмы шифрования с открытым ключом основаны на использовании так называемых односторонних функций. ***Односторонней функцией*** (one-way function)

$$F(X)=Y,$$

называется математическая функция F, значение Y которой относительно легко вычислить для любого входного X, но трудно найти по bpdtcnyjve значению функции Y соответствующее значение аргумента X (т.е. такое, что $F(X)=Y$). Говоря более формально, если для вычисления Y по X существует алгоритм ***полиномиальной сложности***, а для определения X по Y, т.е. известны только алгоритмы ***экспоненциальной сложности***.

Полиномиальным называется такой алгоритм, у которого число необходимых операций можно записать в виде полинома (20) произвольной степени k, где n – число бит входных данных (в нашем случае – число бит в бинарной записи X), a – произвольный множитель.

$$a_K n^K + a_{K-1} n^{K-1} \ldots \quad (20)$$

Алгоритмом экспоненциальной сложности называется такой алгоритм, у которого число необходимых операций пропорционально ak^n , где $k>1$ и a – произвольный множитель. В данном случае при увеличении объема входных данных всего на 1 бит – число необходимых операций растет в 2 раза! Примером экспоненциальной сложности является задача подбора методом «brute force» n-битного ключа шифрования.

Примеры односторонних функций:

1. Задача разложения на простые множители (***факторизация***)

$$N = p \cdot q, \quad (21)$$

где p и q - простые числа.

Разложить большое целое число N на неизвестные простые множители - сложно, однако проверить, перемножив полученные p и q -легко.

2. Дискретное логарифмирование:

$$Y = X^e mod\ N. \quad (22)$$

Задача дискретного логарифмирования: Зная Y, e и N весьма сложно определить X. Зная Y, X и N весьма сложно определить e. Однако, зная X, e и N - легко определить Y.

7.2 Односторонние функции с секретом

Использовать односторонние функции для шифрования сообщений с целью их защиты не имеет смысла, так как обратно расшифровать зашифрованное сообщение уже не получится – экспоненциальная сложность. Процедура расшифровки будет иметь сложность прямого перебора всех возможных вариантов открытого сообщения. Поэтому, для целей шифрования используются специальные односторонние функции – ***односторонние функции*** **с** ***секретом*** – это особый вид односторонних функций, имеющих некоторый дополнительный параметр, позволяющий , при известном значении этого параметра, относительно быстро вычислить обратное значение функции.

$$\mathrm{F}^{-1}(\mathrm{Y}, \mathrm{S}) = \mathrm{X}. \quad (23)$$

То есть, существует некоторая важная информация S , которую можно подставить в формулу (23) вычисления обратной функции, тогда обратная функция будет найдена легко.

Таким образом, объединяя данное свойство со свойствами простых односторонних функций, получим следующее. Для дносторонней функции с секретом $F(X)$ справедливы следующие утверждения:

1. зная X, легко вычислить $F(X)$;
2. по известному значению $Y = F(X)$ трудно найти X;

зная дополнительно некоторую секретную информацию, можно легко вычислить $X = \mathrm{F}^{-1}(\mathrm{Y}, \mathrm{S})$.

Пример: зная Y,e, N необходимо получить X .

$$Y = X^e \bmod N,$$

В общем случае, алгоритм решения этой задачи имеет экспоненциальную сложность. Но, если известно такое значение d, что

$$(ed) \bmod (\varphi(N)) = 1,$$

где $\varphi(N)$ – так называемая функция Эйлера, то по теореме Эйлера:

$$(X^{ed}) \bmod N = Y^d \bmod N = X.$$

Следовательно, чтобы найти по Y,e, N значение X , *нужно просто вычислить* $Y^d \bmod N = X$. В данном случае значение d является тем секретом, которое позволяет сделать решение обратной задачи простым.

7.3Алгоритм шифрования RSA

Разработан в 1977 году в Массачусетском технологическом институте (США). Получил название по первым буквам фамилий авторов (Rivest, Shamir, Adleman).

Алгоритм RSA – алгоритм шифрования с открытым ключом. Традиционная схема распределения ключей в данном случае – как можно более широко распространять открытый ключ (ключ для зашифрования), а закрытый хранить у хозяина.

Алгоритм RSA основан на использовании того факта, что задача факторизации является трудной, т.е. легко перемножить два числа, в то время как не существует полиномиального алгоритма нахождения простых сомножителей большого числа. Алгоритм RSA представляет собой блочный алгоритм шифрования, где зашифрованные и незашифрованные данные должны быть представлены в виде целых чисел между 0 и $N-1$ для некоторого заранее заданного в параметрах шифра большого числа N.

Шифрование RSA делится на 3 этапа:

1) формирование пары ключей

   - выбираем два больших целых числа примерно одного порядка (p,q – простые числа) и вычисляем

   $$N = p \cdot q$$

   Далее вычисляем функцию Эйлера для выбранных простых чисел

   $$\varphi(N) = (p-1)(q-1), \quad (22)$$

   -Далее выбираем целое число e, взаимно простое с $\varphi(N)$, удовлетворяющее условию $e \leq \varphi(N)$,

   Далее вычисляем d: $(e \cdot d) mod \varphi(N) = 1$,

   В результате получаем пару чисел (e, N) – открытый ключ (ключ зашифрования) ; пару чисел (d, N) – закрытый ключ.

Первый этап выполняется единожды, нет необходимости заново его повторять при каждой операции шифрования.

2) зашифрование выполняется с использованием только открытого ключа

$$C = M^e mod N, \quad (23)$$

где M- шифруемое сообщение, представленное в виде большого целого числа, $M \in [0; N-1]$; C – зашифрованное сообщение, $C \in [0; N-1]$.

3) операция расшифрования проводится с использованием закрытого ключа:

$$M = C^d \, mod N. \quad (24)$$

Покажем, что операции зашифрования (23) и расшифрования (24) совместно образуют корректный шифр, то есть для произвольного сообщения покажем, что зашифрование открытым ключем и последующее расшифрование соответсвующим закрытым ключем дает в результате исходное сообщение. Запишем подряд операции зашифрования /расшифрования:

$$(M^e)^d mod N = M^{ed} mod N = M^{(k \cdot \varphi(N)+1)} mod N, \quad (25)$$

где M^e- шифр.

$$(M \cdot M^{k \cdot \varphi(N)} mod N = M \cdot 1^k mod N = M, \quad (26)$$

где $k > 0, k \in Z$

Для преобразования формулы (26) была использована теорема Эйлера, уже упомянутая ранее:

$$M^{\varphi(N)} mod N = 1, \big(M, \varphi(N)\big) = 1, \qquad (27)$$

$M^e mod N$ – базовая односторонняя функция с секретом, которая лежит в основе шифра.

Если число d не известно, то атакующий для дешифрования сообщения, то есть для нахождения неизвестного M , должен решить вычислительно сложную обратную задачу к (23).

Секрет в данном случае - число d, т.е. закрытый ключ.

Что нужно сделать атакующему, чтобы зная основные параметры шифра и открытый ключ вычислить закрытый ключ d на основе открытого ключа e? Для этого атакующему необходимо воспользоваться формулой $(ed) mod\ \varphi(N) = 1$, в которой ему неизвестно значение ф-ции Эйлера $\varphi(N)$, но известно N. *Функцию Эйлера легко вычслить по формуле*

$$\varphi(N) = (p-1)(q-1)$$

но при этом необходимо знать значения простых чисел p и q такие, что $N = p \cdot q$. Фактически, задача вычисления закрытого ключа на основе известного открытого свелать к решению задачи факторизации (т.е. квычислительно сложной задаче - см. примеры односторонних функций).

Таким образом, надежность и устойчивость к взлому шифра RSA полностью зависит от свойств односторонних функций, лежащих в его основе. Если хотя бы одна из этих двух функций будет взломана, то есть будет найден способ вычислить обратную функцию за полиномиальное время – весь алгоритм шифрования будет ненадежным (появится возможность либо расшифровать данные, зная только открытый ключ, либо восстановить закрытый ключ по открытому).

Недостатки RSA:

1) операции зашифрования/расшифрования требуют значительно больше вычислительных ресурсов и, соответственно, выполняются медленнее, чем при использовании симметричных шифров;
2) алгоритмы с открытым ключом имеют определённые особенности (требуется дополнительные операции, чтобы представить шифруемые данные в виде набора целых чисел $\in [0; N-1]$, менее устойчив к взлому), которые затрудняют их использование и делают нежелательным применение этих алгоритмов для зашифрования больших объёмов избыточных данных.

Вывод: для шифрования больших объемов данные RSA не используются. На данный момент нет алгоритмов с открытым ключом, которые подходят для шифрования больших объемов реальных данных (от нескольких мегабайт). Оптимальным объемом данных для RSA – пара килобайт. На практике RSA пригодны для, так называемого ***гибридного шифрования***.

7.4 Гибридное шифрование

Шифры с открытым ключом, тем не менее, эффективны при распространении ключей симметричных шифров и именно для этой цели они используются в ***гибридных криптосистемах***. Гибридный шифр использует и симметричный шифр, и шифр с открытым ключом. Сначала генерируется случайный ключ для симметричного шифра, называемый ***сеансовым ключом***. Сообщение зашифровывается симметричным шифром с использованием сеансового ключа. Затем сеансовый ключ зашифровывается открытым ключом получателя. Сеансовый ключ, зашифрованный шифром с открытым ключом, и сообщение, зашифрованное симметричным шифром, объединяются и передаются получателю по открытому каналу связи. Получатель использует свой секретный ключ для расшифровки сеансового ключа и затем использует полученный сеансовый ключ для расшифровки сообщения. Так как ключ симметричного шифра передаётся защищённым образом, то для каждого сообщения может быть сгенерирован новый сеансовый ключ (таким образом будет обеспечена большая, чем с лучае с многоразовым ключом шифрования, стойкость используемого блочного шифра).

Скорость и надежность на больших объемах избыточных шифру передает используемый блочный шифр, а RSA – дает возможность использовать открытый закрытый ключи. Следует помнить, что стойкость гибридного шифра к взлому определяется стойкостью «самого слабого звена», т.е. достаточно взломать либо симметричный шифр с сеансовым ключом, любо шифр с открытым ключом, чтобы вся схема гибридного шифрования была взломана.

7.5 Алгоритм согласования секретных ключей Диффи–Хэллмана

Алгоритм Диффи – Хэллмана - алгоритм, позволяющий двум и более сторонам получить общий секретный ключ (например, ключ шифрования для симметричного алгоритма ГОСТ 28147-89), используя незащищенный от прослушивания канал связи.

Предположим, существует два абонента: A и B. Обоим абонентам известны некоторые два числа N и X, которые не являются секретными и могут быть известны также другим заинтересованным лицам. Для того, чтобы создать не известный более никому секретный ключ, оба абонента генерируют большие положительные целые случайные числа: A — число a, B— число b. Затем абонент A вычисляет значение

$$K_A = X^a mod\ N, \quad (28),$$

и пересылает (28) абоненту B. Аналогично, абонент B вычисляет

$$K_B = X^b mod\ N, \quad (29),$$

и пересылает (29) абоненту A. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (то есть у него нет возможности вмешаться в процесс передачи). На втором этапе A на основе имеющегося у него a и полученного по сети b вычисляет значение

$$K = (K_B)^a = X^{ab} mod\ N,\ (30)$$

аналогично абонент *B* вычисляет значение

$$K = (K_A)^b = X^{ab} mod\ N.\ (31)$$

Согласно уравнениям (30) и (31), у *A* и *B* получилось одно и то же число *K*. Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник встретится с практически неразрешимой (за разумное время) проблемой вычисления дискретного логарифма (злоумышленнику для вычbсления общего ключа *K* нужно вычислить число *a* или *b*, т.к. эти числа не передавались по открытому каналу связи). Алгоритм надежен только в случаи пассивного атакующего (т.е. только читающего сообщение, но не способного подменить или исказить его). Если атакующий может заменять и удалять сообщение из криптосистемы, то данный алгоритм становится уязвим и подвержен атаке «Человек посередине» (man-in-the-middle).

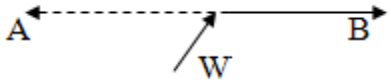


Рисунок 30. Схема атаки «человек посередине» (man-in-the-middle)

Рассмотрим подробнее, как проходит атака. Атакующий *W подменяет передаваемые по каналу связи* K_A *и* K_B на собственное значение K_W. После этого *A и B* генерируют разные «общие» ключи *K и K'*, что по факту делает невозможным обмен зашифрованными сообщениями между ними. В тоже время, у атакующего есть оба ключа *K и K'*, т.е. он может читать все передаваемые сообщения, перешифровывать их нужным ключом и передавать дальше. Таким образом, он будет имитировать полноценную зашифрованную связь между A и B, которые, в свою очередь, никак не смогут обнаружить «посредника», читающего их зашифрованные сообщения.

A	W	B
a	w	b
$K_A = X^a mod\ N$	$K_W = X^w mod\ N$	$K_B = X^b mod\ N$
K_A ⟷	K_W K_W	⟷ K_B
Обмен частичными ключами		
$K = (K_W)^a mod\ N = X^{wa}$	$K = (K_A)^w mod\ N$ $K' = (K_W)^b mod\ N$	$K' = (K_w)^b mod\ N = X^{wb}$

Рисунок 31. Схема атаки «Человек посередине»

Алгоритм Диффи – Хэллмана в чистом виде сложных системах не используется. Алгоритм уязвим, когда активный атакующий находится посередине канала связи, и эта уязвимость может быть ликвидирована либо путем ввода дополнительной проверочной информации, которую знают только абоненты A и B (фактически, нужен дополнительный секретный ключ и протокол не имеет смысла), либо участием в обмене доверенного третьего лица, которое может удостоверить что сообщения от A и B являются подлинными.

7.6 Электронная цифровая подпись

Электронная цифровая подпись (ЭЦП)— реквизит электронного документа, предназначенный для удостоверения источника данных и защиты данного электронного документа от подделки и модификации. Проверка ЭЦП – алгоритм, позволяющий получателю однозначно определить отправителя этого сообщения и избежать подлога со стороны активного атакующего. ЭЦП формирует отправитель, а проверяет получатель.

В алгоритме ЭЦП должны быть две базовые процедуры:

1) Процедура простановки подписи:

$$sign_{K_1}(M) = S, \quad (32)$$

где S- цифровая подпись, M – сообщение;
На входе подписанное сообщение, на выходе ЭЦП в виде некоторого сообщения.

2) Проверить подпись на корректность:

$$Verify_{K_2}(M, S) = True/False. \quad (33)$$

Для того, чтобы отправитель однозначно был идентифицирован и ЭЦП нельзя было подделать, в $sign$ появляется ключ K_1 – по аналогии с асимметричными шифрами этот ключ будет являться закрытым ключом, т.е. этот ключ будет известен только подписывающему. При этом

$$sign_{K_1}(M) = S, sign_{K'_1}(M) \neq S, \text{ для любого } K'_1 \neq K_1 \quad (35)$$

В $Verify$ также появляется открытый ключ, необходимый для проверки ЭЦП. Если K_1 - закрытый ключ для формирования подписи, то K_2 - открытый ключ, предназначенный исключительно для проверки подписи. K_1 и K_2 образуют уникальную пару ключей, K_1 есть только у автора, K_2 присутствует у максимально широкого круга получателей. Если любой проверяющий с K_2, проверяет ЭЦП и прешел к выводу, что она корректна, то автоматически это означает, что ЭЦП сформировано ключом K_1.

7.7 Алгоритм ЭЦП RSA

Рассмотрим применение алгоритма RSA для формирования ЭЦП на примере вычисления и проверки электронной подписи (S) сообщения M. Первый шаг - вычисление хэша сообщения $\mathrm{m} = \mathrm{h(M)}$, который затем шифруется на секретном ключе K_1. Для алгоритма ЭЦП RSA

$$\mathrm{S} = m^{\mathrm{K_1}} \bmod \mathrm{N}. \quad (36)$$

Получатель, желающий проверить значение S сообщения M, также вычисляет хэш сообщения по формуле $m = h(M)$, и расшифровывает S с помощью открытого ключа K_2, используя асимметричный алгоритм шифрования RSA, согласно выражению

$$\mathrm{m'} = \mathrm{S^{K_2}} \bmod \mathrm{N}. \quad (37)$$

Если $m' = m$, ЭЦП сообщения признается верной. В противном случае подпись считается поддельной и делается вывод о том, что целостность сообщения нарушена.

Итак, в криптосистеме RSA секретный ключ используется для вычисления ЭЦП или для расшифрования сообщений, а открытый - для проверки ЭЦП или зашифрования сообщений.

Следует отметить и ряд недостатков, свойственных формированию ЭЦП с использованием RSA, причем часть из них унаследованы от используемого алгоритма шифрования RSA. Среди последних стоит упомянуть о том, что ЭЦП RSA уязвима к мультипликативной атаке, т. е. алгоритм ЭЦП RSA позволяет злоумышленнику, даже не зная секретный ключ K_1, вычислить подписи сообщений, результат хэширования которых совпадает с произведением результатов хэширования подписанных ранее сообщений.

7.8 Атака «человек посередине» на схеме ЭЦП и шифрование с открытым ключом

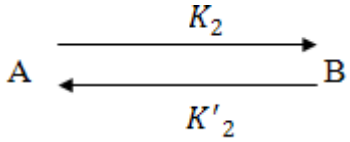


Рисунок 32 . Схема шифрования с открытым ключом

$$E' = M^{K'_2} mod\ N, \quad (38)$$

$$E = M^{K_2} mod\ N. \quad (39)$$

В схеме шифрования с открытым ключом в первую очередь получатель и отправитель зашифрованного сообщения передают друг другу открытые ключи, и лишь затем производится шифрование передаваемых сообщений (рис. 32). Но в случае, когда открытый канал связи, по которому передаются ключи, контролируется атакующим (W) , схема обмена данными может выглядеть следующим образом.

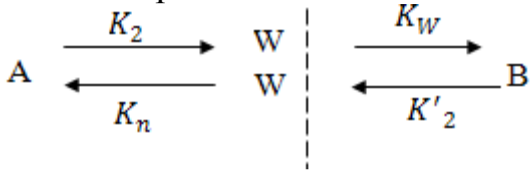


Рисунок 33 – Обмен данными при атаке «человек посередине».

Атакующий подменяет открытые ключи, передаваемые пользователями, на свой собственный открытый ключ. В этом случае, дальнейший обмен зашифрованными сообщениями между A и B осуществляется по схеме:

$$E' = M^{W} mod\ N, \quad (40)$$

Фактически,и A, и B шифруют передаваемые сообщения одним и тем же ключом атакующего, но при этом не могут выявить факт атаки. Как и в случае с атакой на алгорит Диффи-Хеллмана, атакующий в этом случае может перехватывать и расшифровывать все передаваемые сообщений, сам при этом оставаясь необнаруженным (у атакующегов этой ситуации есть оба открытых ключа пользователей A и B, что позволяет ему «на лету» перешифровывать полученные данные нужным ключом)

Способ решения проблемы – использование инфраструктуры открытых ключей.

7.9 Инфраструктура открытых ключей

Инфраструктура открытых ключей (англ. ***PKI*** - *Public Key Infrastructure*) - технология аутентификации с помощью открытых ключей, основанная на существовании общего для всех пользователей «арбитра», т.е. доверенного центра подписи.

Основные механизмы PKI:

- установление доверия (в рамках заданной модели доверия)
- система именования субъектов, обеспечивающая уникальность имени в рамках системы
- связь имени субъекта и пары ключей (открытый и закрытый) с подтверждением этой связи средствами удостоверяющего центра, которому доверяет субъект, проверяющий правильность связи

PKI не реализует авторизацию, доверие, именование субъектов криптографии, защиту информации или линий связи, но может использоваться как одна из составляющих при их реализации.

PKI также может использоваться для реализации конфиденциальности, целостности передаваемых данных, неотказуемости (или апеллируемости - англ. non-repudiation).

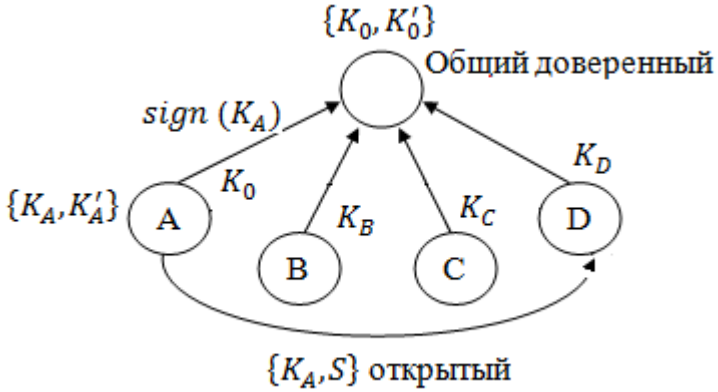


Рисунок 34. Схема обмена открытыми ключами с участием арбитра

где K_A - открытый ключ, K_A' - закрытый ключ.

Отдельный пользователь не может просто сгенерировать и сразу использовать открытый ключ.

Этапы:

1) Абонент A отправляет K_A арбитру по защищенному каналу связи. Защищенный канал – канал связи, в котором сообщение не подвержено ни искажению, ни подмене;
2) У арбитра есть пара $\{K_0, K_0'\}$. Арбитр обратно отправляет A его открытый ключ, подписанный его (арбитра) закрытым ключом;
3) Если A хочет установить связь с D, то он отправляет ему свой открытый ключ совместно с подписью, сформированной арбитром, при этом передача идет по открытому каналу связи; атака «человек посередине» в этом случае нереализуема из-за подписи арбитра;

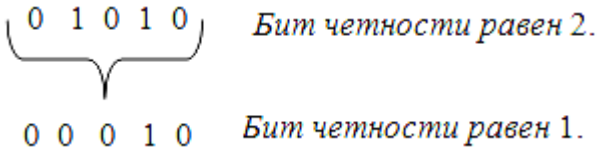
4) Ключ K_0 для проверки подписи арбитра рассылается арбитром всем пользователям по защищенному каналу связи. Пользователь D, получив открытый ключ от A, проводит процедуру проверки подписи арбитра $Verify_{K_0}(K_A, S)$ и удостоверяется, что при передачи по открытому каналу ключ пользователя A не был подменен «человеком посередине».

ЛЕКЦИЯ 8. ХЕШ-ФУНКЦИИ, КОДЫ АУТЕНТИФИКАЦИИ СООБЩЕНИЙ (MESSAGE AUTHENTICATION CODE)

8.1 Криптографический алгоритм контроля целостности

Криптографические методы позволяют надежно контролировать целостность как отдельных порций данных, так и их наборов (таких как поток сообщений); определять подлинность источника данных. В отличие от электронной цифровой подписи, большинство алгоритмов данного типа основаны на общем секретном ключе и не позволяют обеспечить "неотказуемость" при подписывании сообщений (невозможность отказаться от совершенных действий).

Примером простейшего некриптографического алгоритма контроля целостности является алгоритм вычисления бита четности: сложить все биты сообщения по mod2 и получить «проверочный» бит:



Биты четности не могут выявить факт перехвата сообщения злоумышленником, его изменения и последующей отправки получателю, т.к. атакующий, зная алгоритм расчета бита четности, может преднамеренно скорректировать свое сообщение таким образом, чтобы бит четности совпадал с битом четности подлинного сообщения. При этом, в случае бита четности и других некриптографических алгоритмов, существует простой способ расчета, позволяющий атакующему сформировать произвольное сообщение с нужным битом четности.

Для защиты от преднамеренного искажения необходимы алгоритмы , позволяющие успешно выявлять факты и умышленного, и неумышленного изменения данных.

8.2 Криптографические хеш-функции

Хеширование (иногда хэширование, англ. *hashing*) — преобразование входного массива данных произвольной длины в выходную битовую строку фиксированного размера. Такие преобразования также называются ***хеш-функциями.*** Изменение хотя бы 1 бита входных данных (хешируемого сообщения) приводит к значительным изменениям выходных данных, что может быть обнаружено получателем сообщения при попытке повторного вычисления хш-функции от полученных данных.

$$HASH\ (M) = H. \quad (42)$$

Для того, чтобы хеш-функция H считалась криптографически стойкой, она должна удовлетворять двум основным требованиям, на которых основано большинство применений хеш-функций в криптографии:

1) Зная H, нельзя восстановить сообщение M или его часть – ***свойство необратимости*** (более подробно см. односторонние функции);
2) ***Стойкость к коллизиям***.

Коллизией хеш-функции называется ситуация, когда для двух различных сообщений

$$HASH\ (M) = HASH\ (M').\quad (43)$$

Существует два типа коллизий типов:

1. Коллизия 1-го типа: возможность (за полиномиальное время) подобрать пару сообщений M и M', чтобы для них выполнялось равенство (43);
2. Коллизия 2-го типа: возможность, зная конкретное значение хеш-функции H, за полиноминальное время подобрать значение M, чтобы выполнялось условие (42).

. Пусть результатом хеш-функции будет битовый вектор фиксированной длины n, который позволяет проверить целостность сообщения. Общее количество таких векторов можно определить, как 2^n . Любая хэш-функция, принимающая на вход сообщения произвольной длины и выдающая результат фиксированной длины, будет иметь коллизии. Стойкость к коллизиям в данном случае означает, что атакующему нужно перебирать в среднем $\frac{2^n}{2}$ разных сообщений, чтобы столкнуться с коллизией хеш-функциями. Другими словами, у стойких хеш-функций коллизии есть, но чтобы их найти, нужно решать экспоненциально сложную задачу перебора разных входных сообщений.

Пример: пусть задано сообщение в 1000 бит, $n = 121 - $ длина выхода хеш $-$ функции в битах . Необходимо определить количество сообщений с одинаковым $HASH$.

2^{1000} – общее число возможных сообщений;

2^{121} – возможное число $HASH$;

тогда по формуле $\frac{2^{1000}}{2^{121}} = 2^{872}$- сообщений с одинаковым $HASH$. Любые из этих 2^{872} сообщений образуют коллизию

Похоже на одностороннюю функцию, хеш должен быть необратим, устойчив к коллизиям.

8.3 Односторонняя функция со сжатием

Односторонняя функция со сжатием – функция, на основе которой обычно строятся хеш-функции. Односторонняя функция со сжатием должна обладать свойствами стойкости к коллизиям и необратимости, но при этом не требуется работа с произвольной длиной входного сообщения. Односторонняя функция со сжатием берет на вход *2n* бит и выдает результат *n* бит (рис. 35).

В качестве примера такой функции можно рассмотреть побитовый XOR – пусть первые n бит входного сообщения XORятся с последними n битами.

К сожалению, при том, что данная функция будет обеспечивать необратимое сжатие, она будет уязвима к коллизиям – дейсвительно, легко подобрать множество сообщений с одинаковых результатом XOR.

Рисунок 35. Односторонняя функция со сжатием – пример.

Примером стойкой к коллизиям односторонней функции со сжатием является функция

$$Y = X_1^{X_2} mod\ N,$$

где X_1 и X_2 – две половины сообщения , N – большое целое число, разрядность в битах которого превышает X_1 и X_2

8.4 Итеративная схема построения хеш-функции.

Данная схема позволяет построить хеш-функцию из готовой односторонней функции со сжатием. Общий принцип – итеративно применять одностороннюю функцию к блокам сообщения, число итераций при этом равно числу блоков. В качестве односторонней функции со сжатием часто используется блочный шифр - $Y = E_{X_1}(X_2)$.

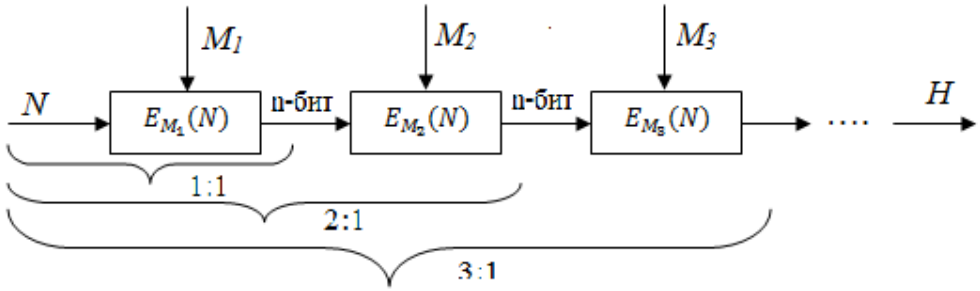


Рисунок 36. Итеративная схема построения хеш-функции

Разбиваем сообщение M на блоки длиной n бит каждый: M_1, M_2, M_3 . Эта схема позволяет сделать хеш-функцию. N – вектор инициализации длиной n бит, случайное число; односторонняя функция со сжатием заключается в шифровании этого вектора с ключом M_1 . По итогам первой итерации реального сжатия не достигнуто – после шифрования n-битным блоком M_1 на выходе получен тоже n-битный блок. Степень сжатия входящего сообщения нарастает по мере увеличения числа итераций – см рис. 36.

Изменение хотя бы 1 бита хотя бы в одном из блоков вызывает значительное изменение в общем результате хеширования, за счет лавинного эффекта шифра и режима «зацепления» блоков M_1, M_2, M_3. IV о должно быть заранее известно для корректного подсчета хеш-функции.

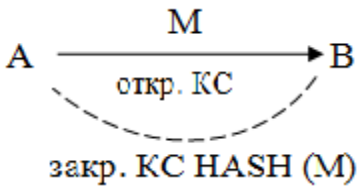


Рисунок 37. Корректная проверка целостности при использовании защищенного канала связи

Чтобы удостоверить целостность, необходимо по защищенному каналу передать хеш *HASH (M)*, а само сообщение – можно передавать по открытому.

Получатель B пересчитает *HASH* от полученного сообщения и сравнит его с тем, что он получил по защищенному каналу

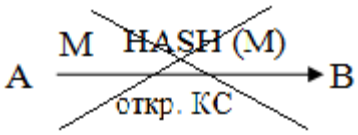


Рисунок 38. Некорректная проверка целостности

Рисунок (38) показывает, что при наличии только открытого канала связи данной схемой пользоваться нельзя, так как злоумышленник в открытом канале заменит и исказит как само сообщение, так и $HASH$ (так, чтобы новое сообщение соответствовало новому хешу).

8.5 Хэш-функция SHA-1 (Документ RFC-3174)

Некоторые параметры:
- размер входного блока равен 512 бит (размер должен быть кратен 512-ти);
- размер выходного хеша равен 160 бит.

Хеш-функция построена на расширенной сети Фейстля. Количество итераций для одного 512-ти битного блока равно **80**.
Схема одной итерации представлена ниже:

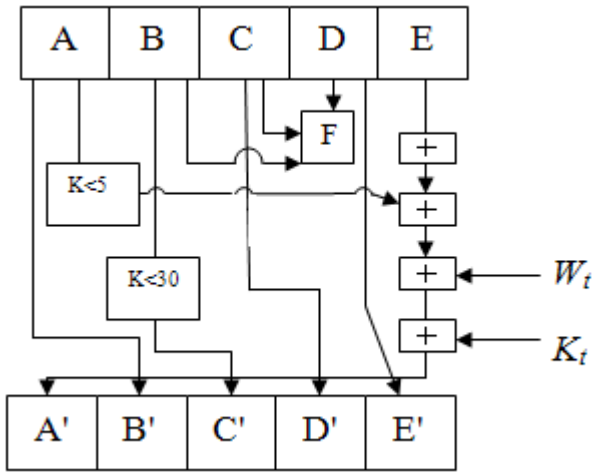


Рисунок 39. Схема одной итерации SHA-1

где $\boxed{+}$ - сложение по (mod 2^{32}), $\boxed{K<5}$ - циклический сдвиг на 5.

A, B, C, D, E – пять 32-битных регистров, при начале вычисления хеш-функции они заполняются заранее известными константами. K_t - аналог ключа в шифрах на основе сети Фейстеля, однако отличие от SHA-1 от шифров в том, что $K_t = const$. W_t – подблок хешируемого 512битного сообщения, на каждой итерации используется новый (размером в 32 бита) подблок из набора $(W_0, W_1 \dots W_{79})$.
W_t- набирается на 512 бит $(32 \cdot 80 \gg 512)$.всего за 16 итерации - $(W_0 \dots W_{15})$!
Где брать W_t для остающихся итераций. В оставшиеся 64 итерации W_t фактически псевдослучайно генерируется на основе предыдущих значений:

$$W_t = W_{t-3} \oplus W_{t-8} \oplus \dots \oplus W_{t-14} \oplus W_{t-16}. \quad (45)$$
$$15 < t < 80$$

Здесь W_t начиная с W_{16} не берутся из хешируемого сообщения. Все оставшиеся W высчитываются по формуле для генератора Фибоначи.

8.6 Алгоритм работы хеш-фнукции SHA-1 с сообщениями разной длины

Сообщение дополняется до размера кратного 512-ти. Далее запускается хеширование первого одного блока, состоящее из 80 итераций этой схемы.
$K = const$, хешируемое сообщение в W. После 80 итераций получилось некоторое заполнение регистров A', B', C', D', E', которые мы сохраняем в H_A, H_B, H_C, H_D, H_E – (временные переменные), результат хеширования первого блока записан во временных переменных. Затем то же самое делаем со вторым блоком, но инициализируем не константами A, B, C, D, E, а результатом хеширования предыдущего блока – $A', B', C', D',$. После второго блока получаем $A'_2, B'_2, C'_2, D'_2, E'_2$, и сохраняем

$$H_A = H_A + A'_2, \quad (47),$$

и аналогично для H_B, H_C, H_D, H_E
После обработки всех блоков H_A, H_B, H_C, H_D, H_E склеиваются в одну строку и являются общим результатом хеширования.

Существует много реализаций SHA-1. На данный момент функция теоретически взломана, но практического подтверждения нет. Так как для подбора необходим хеш 2^{159}. В то время как, наиболее успешная атака требует сейчас 2^{69} операций.

8.7 MAC (Message Authentication Code)

MAC - в протоколах аутентификации сообщений с доверяющими друг другу участниками — специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных.
В отличие от хеш-функций, вMAC появляется свойство зависимости от ключа – т.е. корректно посчитать и проверить значение MAC от сообщения можно,, только зная ключ (в отличие от хеш-функции). В этом случае и сообщение и MAC для этого сообщения могут передаваться по открытому каналу связи. Нужно лишь один раз создать закрытый канал связи, чтобы согласовать пароль между получателем и отправителем. Рассмотрим две конструкции MAC:

1) MAC на основе хеш-функции. Определяем хеш-функцию следующим образом: $\underbrace{HASH\ (msg\ + password)}_{\text{MAC}}$

В качестве $HASH$ берется любая стойкая хеш-функция. В этой схеме есть уязвимость, позволяющая подделать MAC в некоторых специфичных случаях, но она может защитить от большинства атак.

Стандарт для MAC на основе хеш-функции: *HMAC (*RFC 2104*)*.

$$HMAC_K(m) = HMAC((K \oplus opad)\|HASH\ (K \oplus ipad)\|m)).\ (48)$$

$opad$ = O x 5c5c5c…..

$ipad$ = O x 3636….

m- сообщение, которое прогоняется через HASH два раза.

$ipad$ – 0101, $opad$ – 1010 – нужны, чтобы в $(K \oplus opad)$ и в $(K \oplus ipad)$ получить два сильно различающихся (по расстоянию Хэмминга) подключа.

Достоинства: надежен с точки зрения криптографической стойкости;

Недостатки: сложность формулы: для вычисления MAC необходимо два раза вычислить хеш-функцию.

2) MAC на основе блочного шифра + хеш-функции:

$$H = E_K(HASH\ (m)),\ (49)$$

Сначала считается хеш-функция, затем шифруется секретным ключом. Возможно наоборот:

$$H = HASH\ (E_K(m)),\ (50)$$

H и исходное сообщение передаются по открытому каналу связи.

(49) – считается надежной, так как злоумышлений не знает ключ. (50) – аналогично, злоумышленник не знает, от какого сообщения считать хеш-функцию.

Недостаток: чтобы реализовать такую схему надо реализовать и хеш-функцию, и шифрование.

MAC на основе блочного шифра

Алгоритм:

1) Блочным шифром в режиме простой замены шифруем первый блок сообщения *m* с ключом *K*;
2) Результат шифрования первого блока XOR им со вторым блоком, после чего шифруем второй блок и так до последнего блока.

В данном режиме, даже при наличии ключа, исходный текст нельзя расшифровать из MAC, но повторить операцию шифрования над известным текстом и тем самым однозначно проверить MAC сообщения-возможно.

Достоинства: из существующих аналогов это самый быстрый алгоритм. Он стандартизован в ГОСТ 28147-89.

8.7 Шифр ГОСТ 28147-89. Режим формирования имитоприставки.

Пусть известен ключ шифрования *K*, известно хешируемое сообщение *M*. Сообщение *M* разбивается на 64-битные блоки: M_1, M_2 ... M_1, которые шифруются ключом *K* в режиме простой замены. Вместо 32 итераций используем 16. После шифрования первого блока M_1 результат шифрования одного блока делится на 2 подпблока подвергается дополнительной операции: складывается по модулю 2^{32} *с константой C (описана в стандарте). Результат сложения XORится* со вторым блоком M_2, и только после этого результат операции *XOR* зашифровывается как второй блок. Аналогично для 3го и последующих блоков (). Размер имитоприставки ВСЕГДА равен 64 бита,

вне зависимости от числа блоков в исходном сообщении. Если необходима простая хеш-функция, то ключ можно установить равным *const*, но данную константу необходимо распространять вместе с хэшем, чтобы получатель мог проверить его корректность. В целом, не рекомендуется использовать такой алгоритм в качестве хеша, т.к. его надежность проверялась только для случая секретного ключа, а не ключа-константы, как требуется для хеша. В том случае, если только определенные лица имеют право генерировать и проверять имитоприставку, ключ должен храниться в секрете.

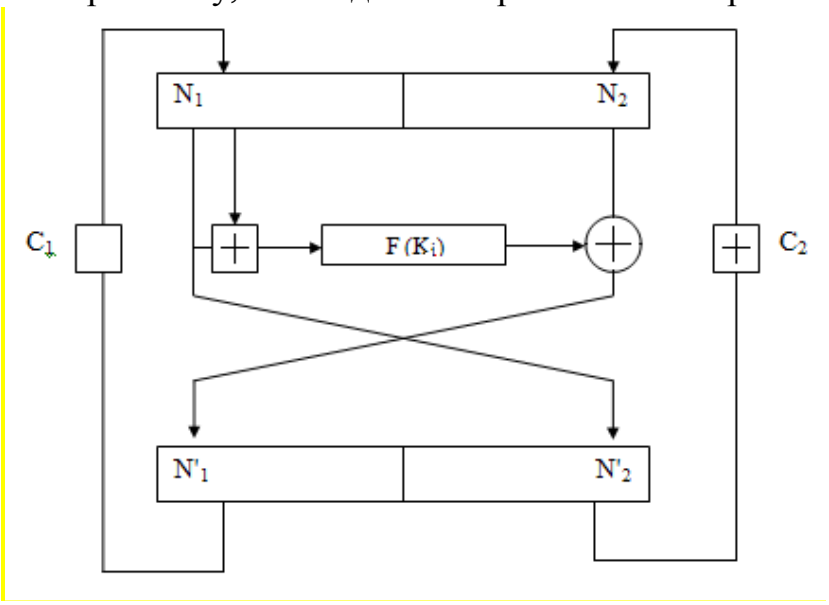


Рисунок 40. Схема алгоритма зашифрования ГОСТ 2847-89 в режиме формирования имитоприставки

ЛЕКЦИЯ 9.ПРОТОКОЛЫ АУТЕНТИФИКАЦИИ

Все протоколы аутентификации включают двух участников:

1) *A* – доказывающего – участника, проходящего аутентификацию и доказывающего участнику В что он «подлинный»;
2) *B* – проверяющего – участника, проверяющего аутентичность доказывающего.

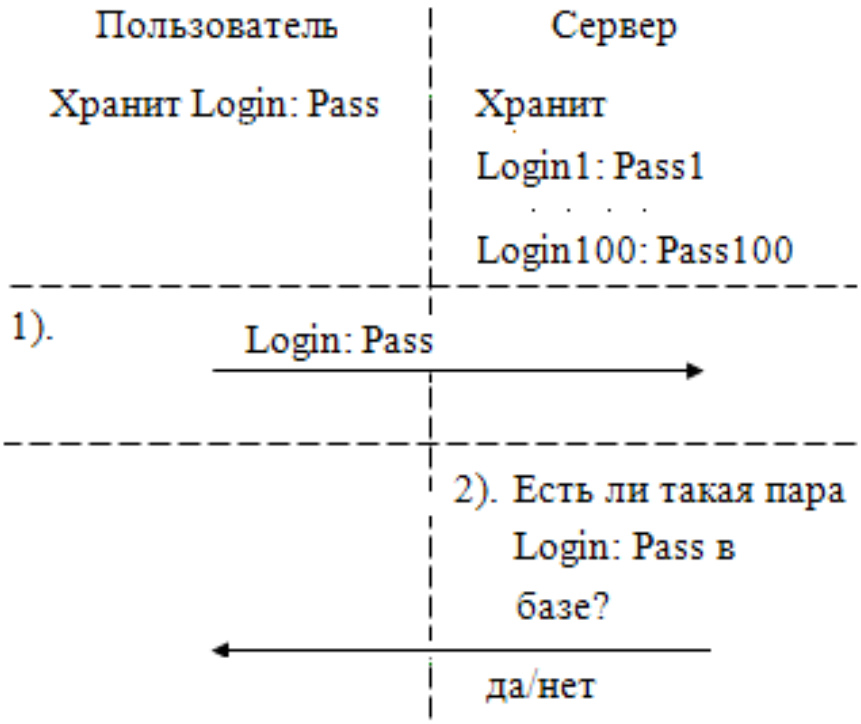
Аутентификация подразумевает активные действия со стороны и проверяющего и проверяемого. Целью протокола является проверка того, что проверяемый действительно является *A*. С точки зрения проверяющего возможными исходами протокола являются либо принятие решения об идентичности доказывающего *A*, либо завершение протокола без принятия такового решения.

Протоколы аутентификации могут быть разбиты на три большие категории в зависимости от того, на чем основана аутентификация:

1) ***«то, что я знаю»*** - на основе знания чего-либо. Примером могут служить стандартные пароли, персональные идентификационные номера (PIN), а также секретные и открытые ключи, знание которых демонстрируется в протоколах типа «запрос-ответ».
2) ***«то, что у меня есть»*** - на основе обладания чем-либо. Обычно это магнитные карты, смарт-карты, touch memory и персональные генераторы, которые используются для создания одноразовых паролей.
3) ***«то, что я есть»*** - на основе каких-либо неотъемлемых характеристик. Эта категория включает методы, базирующиеся на проверке пользовательских биометрических характеристик (голос, сетчатка глаза, отпечатки пальцев).

9.1 Аутентификация со статичной информацией

Данный вид аутентификации предполагает, что для аутентификации каждый раз используется один и тот же общий секрет (пароль, PIN и т.д.). Для каждого пользователя имеется пароль, обычно представляющий собой последовательность знаков алфавита. Эта последовательность выступает в качестве общего секрета пользователя и системы. Для того, чтобы получить доступ к системному ресурсу, пользователь представляет свой идентификатор (*Login* 1) и пароль (*Pass* 1), и прямо или косвенно определяет необходимый ресурс.



Пароли всех пользователей хранятся , по сути, на сервере в открытом виде. При обращении пользователя система сравнивает введенный пароль с паролем данного пользователя, хранимым в файле. Недостаток этого метода состоит в том, что пароли не защищены от несанкционированного доступа к серверу. Для устранения этого недостатка используются зашифрованные файлы паролей пользователей. При этом может использоваться либо непосредственное зашифрование паролей с помощью криптографического алгоритма, либо вычисление значения хэш-функции пароля. Например, в случае с использованием хешей, пользователь передает на сервер пароль, от которого сервер вычисленяет хеш и сравнивает с хешем, хранящимся в его базе данных. При этом, даже получив к базе данных сервера, нельзя простым полиномиальным алгоритмом восстановить исходные пароли из хешей, хранящихся на сервере.

Заметим, что использование шифрования\хеширования перед передачей пароля по открытому каналу связи хотя и защищает сам пароль, но не защищает от возможности вхождения противника в систему путем повторной пересылки ранее перехваченного пароля. Предположим, что после успешной аутентификации клиент отключается от сервера. В то же время, атакующий, ранее перехвативший из канала связи значения $Login\ 1$ и $Pass\ 1$ может пройти аутентификацию, просто переслав серверу эти данные от своего имени.. Таким образом, если аутентификация идет по открытым каналам связи, использование статической информации (пароля или его хеша) для аутентификации всегда дает возможность атакующему осуществить доступ к системе, имитируя законного пользователя.

8.2 Алгоритмы аутентификация с динамической информацией

Повышению надежности аутентификации служит использование ***одноразовых паролей***, то есть паролей, которые могут быть использованы для

аутентификации только единожды. Такие схемы обеспечивают защиту от атакующего, использующего перехват паролей из открытого канала.
Существует несколько схем использования одноразовых паролей. Простейшим вариантом является следующая схема.
Все пользователи системы имеют список паролей, сгенерериванных на сервере и переданных пользователю по защищенному каналу связи. Весь список паролей для каждого пользователя также хранится на сервере. Каждый пользователь, проходя аутентификацию, указывает в качестве пароля один из паролей в списке. Сервер проверяет наличие такого пароля, после чего удаляет этот пароль из своего списка (это не позволит атакующему пройти аутентификацию с перехваченным паролем).

Эта система является надежной, однако у нее есть недостаток – сервер не позволяет хранить таблицу паролей на огромное количество пользователей.

8.2.1Схема Лемпарда (hash chain);

В этой схеме пользователь сам генерирует цепочку одноразовых паролей по следующей схеме. Для выбранного пользователем пароля вычисляется $HASH(Pass), HASH\big(HASH(Pass)\big) \dots HASH_{500}(Pass)$
$HASH_{500}(Pass) -$. исходный пароль к которому последовательно применена хеш-функция (500 раз). При первой регистрации на сервере пользователь передает серверу $P0 = HASH_{500}(Pass)$.

Во время аутентификации, пользователь передает на сервер предыдущий пароль из списка, т.е. $P1 = HASH_{499}(Pass)$. Сервер, зная $HASH_{500}(Pass)$, может проверить что полученный от пользователя P1 действительно удовлетворяет условию *HASH(P1)=P0*. (рис. 41)

После того, как сервер провел проверку и авторизовал пользователя, он сохраняет у себя в базе данных уже $P1 = HASH_{499}(Pass)$. Соответсвенно, при следующем сеансе аутентификации пользователь должен будет прислать уже $P2 = HASH_{498}$.

При этом, если использовалась стойкая хеш-функция, атакующий, перехвативший ранее $HASH_{500}(Pass)$, не может восстановить из него $P1 = HASH_{499}(Pass)$, требуемый для входа в систему.

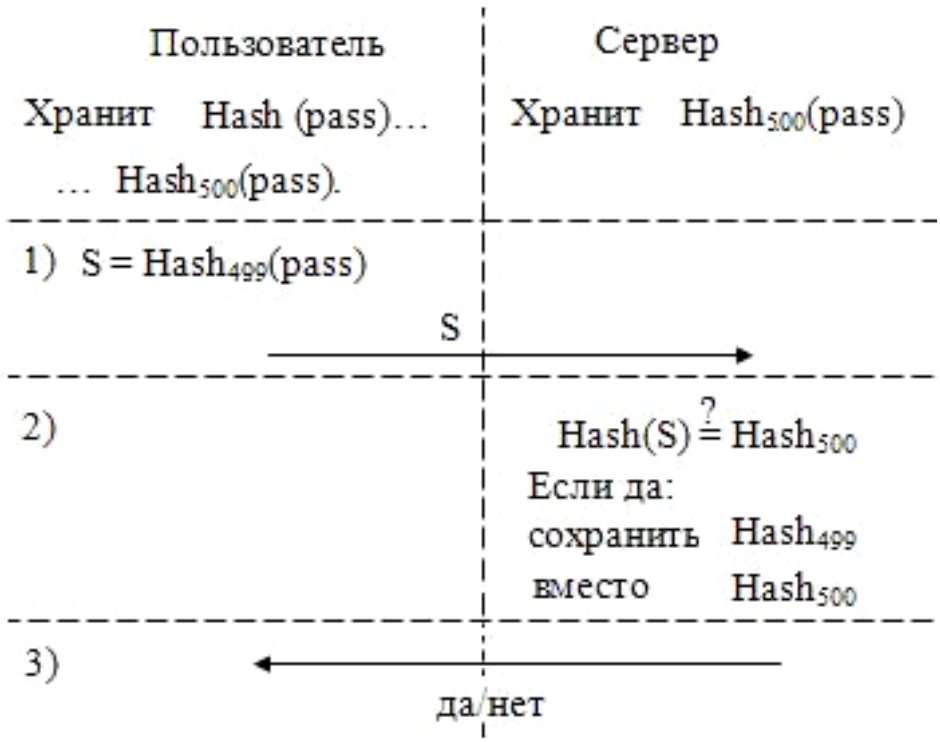


Рисунок 41. Аутентификация по схеме Лэмпарда с одноразовыми паролями.

Достоинством метода является то, что одноразовые пароли генерируются и хранятся у пользователя. **На сервере всегда хранится только одно значение хеша на каждого пользователя**.

Недостатком является то, что пароли должны использоваться пользователем в строгой очередности (500, 499, …). В случае потери хотя бы одного, аутентификацию нельзя будет пройти без ресинхронизации.

Процедура ресинхронизации для схемы Лемпарда:

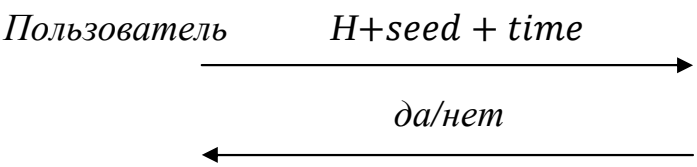
1) Если сервер не опознал пароль, тогда сервер отправляет пользователю значение *HASH*, которое записано у него как последнее актуальное;
2) Пользователь находит направленное значение HASH в списке своих паролей и высылает серверу правильный ответ.

8.2.2 Схема одноразовых паролей с отметкой времени

Метки времени используются для обеспечения гарантий своевременности и единственности сообщений, а также для обнаружения попыток навязывания ранее переданной информации. Они также могут быть использованы для обнаружения попыток задержки информации со стороны атакующего.

Протоколы, использующие метки времени, реализуются следующим образом. Пользователь и сервер знают общий секретный ключ K, при аутентификации пользователь должен вычислить и переслать серверу сложную строку из трех компонентов: значение H согласно (58), отметка времени , $\boldsymbol{seed}$, и передает эту строку серверу по открытому каналу

$$H = E_K(time + seed + pass). \quad (58)$$



где ***seed*** – достаточно длинная случайно сгенерированная строка, определяется сервером. Желательно, но не обязательно, чтобы она регулярно менялась сервером.

Пользователь и сервер предварительно должны согласовать ключ *K*, который шифруется один раз и через защищенный канал связи.

Получив такое сообщение, сервер пытается расшифровать *H* известным ему ключом *K*. Алгоритм аутентификации сервером:

- проверить насколько расшифрованная часть совпала с тем, что отправил пользователь;
- насколько отметка времени, присутствующая в этой строке, отличается от текущего времени на сервере.

На основании данных, полученных в результате проверки, сервер совершает следующие действия:

- в случае совпадения расшифрованного сервером сообщения с тем, что прислал пользователь, а также если его временная метка находится в пределах приемлемого интервала (30 секунд) - сообщение принимается, пользователь прошел аутентификацию;
- иначе – в аутентификации отказано.

Надежность методов, основанных на метке времени, зависит от надежности и точности синхронизации системных часов, что является главной проблемой в таких системах. Преимуществом указанных систем является меньшее число передаваемых для аутентификации сообщений, а также отсутствие требований по сохранению информации для каждой пары участников.

8.2.3 Протокол с аутентификацией «запрос-ответ»

Идея построения криптографических протоколов аутентификации типа «запрос-ответ» состоит в том, что доказывающий убеждает проверяющего в своей аутентичности путем демонстрации своего знания некоторого секрета без прямого предъявления самого секрета.

Рассмотрим несколько ситуаций:

1 Запрос- ответ на основе симметричного шифрования:

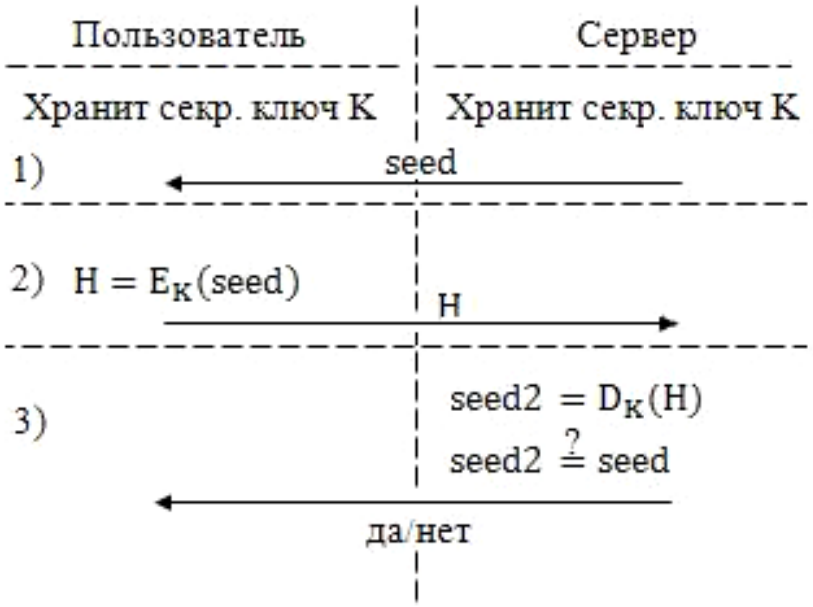


Рисунок 42 – Протокол аутентификации «запрос-ответ»

Принцип работы протокола следующий: в начале процедуры аутентификации пользователь отправляет на сервер свой логин. В ответ на это последний генерирует некую случайную строку *seed* и посылает ее обратно. Пользователь с помощью своего ключа зашифровывает эти данные $H = E_K(seed)$ и отправляет их назад. Сервер же в это время «находит» в своей памяти секретный ключ данного пользователя и вычисляет с его помощью исходную строку.

$$seed2 = D_K(H). \quad (59)$$

Далее проводится сравнение обоих результатов ($seed2 == seed?$). При их полном совпадении считается, что аутентификация прошла успешно.

В отличие от предыдущего протокола, $seed$ должен заново генерироваться сервером для каждого сеанса аутентификации.

Этот метод реализации технологии одноразовых паролей, в отличие от всех остальных, называется асинхронным, поскольку процесс аутентификации не зависит от истории работы пользователя с сервером, порядка предъявления одноразовых паролей и других факторов. Вместо отметки времени выступает интервал timeout между отсылкой сервером seed и получением сервером *H*.

8.2.4 Запрос-ответ на основе ЭЦП.

возникает ситуация, когда нет возможности согласовать секретный ключ *K*, но при этом у сервера есть открытый ключ, гарантированно принадлежащий соответсвующему пользователю, а у пользователя –открытый и закрытые ключи.

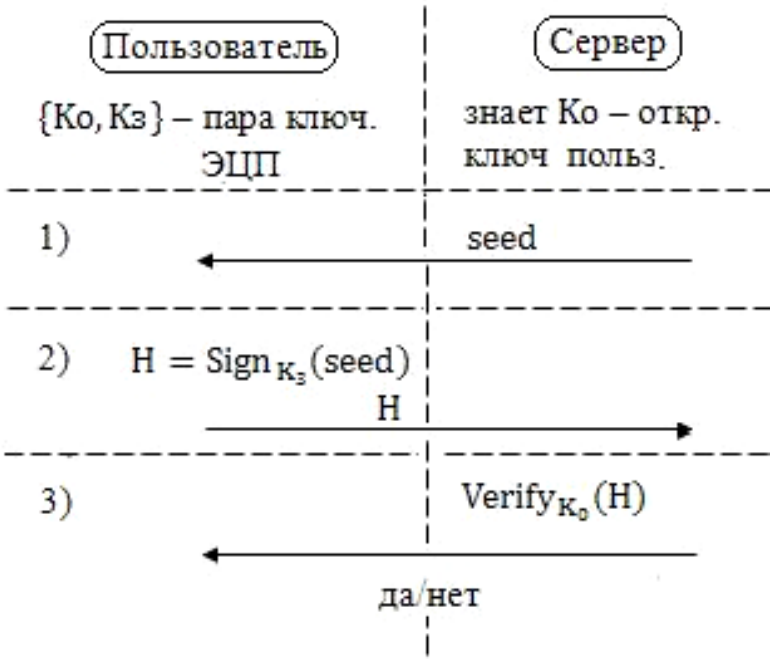


Рисунок 43 – Протокол аутентификации «Запрос-Ответ» на основе ЭЦП

В данной ситуации не требуется защищенный канал связи для согласования общего секретного ключа.

Сервер направляет пользователю *seed*, пользователь подписывает эту строку, затем отправляет полученную ЭЦП серверу. Сервер с помощью открытого ключа пользователя проверяет ЭЦП.

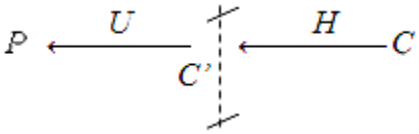
Недостатки:

- возможны злонамеренные действия сервера. Сервер может выбираемое им значение *salt* прислать не случайно, а направить осмысленную строку, например поддельное сообщение третьей стороне якобы от данного пользователя.
- атака «человек посередине», то есть сервер – не тот, за кого себя выдает.

$$P \xleftarrow{salt} C' \xleftarrow{salt} C$$

где *C'* – «человек посередине», ложный сервер аутентификации, который пытается пройти аутентификацию на подлинном сервере С и выдать себя за пользователя *P*.

C' представляется пользователю *P* настоящим, пересылает пользователю *salt*, а *P* не способен отличить *C'* и *C*.



где *C'* получает сообщение и разрывает связь, затем пересылает подмененное *H* настоящему серверу. Если время меньше timeout, то *C'* пройдет аутентификацию вместо настоящего пользователя.

Этой атаке подвержены все схемы запрос-ответ, кроме случаев т.н. «двусторонней аутентификации».

8.3 Схема разделения секретов

Применяется, когда необходимо важную информацию (например, ключ шифрования) разделить между несколькими пользователями таким образом,

чтобы содержание данной важной информации можно было восстановит ьтолько совместными действиями ВСЕХ участников. Пусть требуется разделить на S пользователей ключ (пароль) шифрования $Pass$, состоящий из 5ти символов:

$$Pass = "\_\_\_\_\_"$$

Обозначим пользователей, между которыми будет разделен пароль, как $U_1, U_2, U_3, \dots, U_S$.

Самым тривиальным способом разделения является посимвольный способ – т.е. каждому пользователю сообщается только один символ пароля. Но данный способ имеет существенный недостаток. Хотя формально четверо из пяти участников и не знают всю информации, но им, если они объединят свои усилия, будет достоверно известно 4 символа из 5ти и оставшийся неизвестным символ они смогут подобрать за достаточно малое время. Аналогично, в сговор могут вступить двое пользователей, трое – и в каждом из этих случаев им будет доступна значительная информация о части $Pass$ (соответственно, будут известны 2 или 3 символа из 5ти). Таким образом, при нарастании количества сговорившихся пользователей сложность подбора правильного $Pass$ снижается и, тем самым , вся схема разделения пароля является уязвимой к сговору пользователей.

Рассмотрим схему разделения секрета, которая устойчива к сговору пользователей. В данном случае возможны только два варианта узнать $Pass$ –

- Если объединяются все S пользователей, то $Pass$ становится им известен без каких-либо вычислительно сложных операций (подбора, вычисления сложных функций и т.д.)
- Если вступают в сговор от 2 до *S-1* пользователей, то $Pass$ становится им известен только путем полного перебора (перебор по всем 5ти символам). Ни один символ секрета $Pass$ при этом не является достоверно известным.

Стойкая к сговору схема разделения секрета на S пользователей будет в простейшем виде выглядеть следующим образом:

Для первых *S-1* участников в качестве их части секрета используется не какая-либо часть секрета $Pass$, а генерируется случайные битовые строки, никак не связанные с секретом $Pass$, но имеющие равную ему длину, то есть

$$rand\ 1 = "\_\_\_\_\_"$$
$$rand\ 2 = "\_\_\_\_\_"$$
$$\dots$$

Только для S-го пользователя его часть секрета не генерирется, авычисляется следующим образом:

$$rand5 = pass \oplus rand1 \oplus \dots \oplus rand4,$$

(здесь $\oplus$ $-$побитовая операция XOR).

Для восстановления секрета пользователям достаточно вычислить следующее выражение

$$pass = rand1 \oplus \dots \oplus rand5.$$

Стойкость данной схемы, фактически, основана на известных свойствах «идеального шифра», в котором используется побитовый XOR и

случайно сгенерированный ключ, длина которого равна длине сообщения. Фактически, при сговоре любого числа пользователей, для нахождения общего секрета им потребуется решить задачу взлома идеального шифра – т.е., по факту, перебрать все возможные значения ключа (или все возможные значения общего секрета, что равнозначно по сложности). При этом, в силу свойств идеального шифра, сговор пользователей не дает им никакой дополнительной информации об общем секрете или его части.
\

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Кафедра геоинформатики и информационной безопасности

КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Методические указания к лабораторным работам 1-3

Самара 2013

Цель работы – изучение простейших шифров подстановки и перестановки; получение навыков программной реализации алгоритмов шифрования.

1. Теоретические основы лабораторной работы.

1.1. Простейшие (одноалфавитные) коды подстановки.

Криптография — тайнопись. Термин ввел *Д. Валлис.* Потребность шифровать и передавать шифрованные сообщения возникла очень давно. Так, еще в V-IV вв. до н. э. *греки* применяли специальное шифрующее устройство. По описанию *Плутарха,* оно состояло из двух палок одинаковой длины и толщины. Одну оставляли себе, а другую отдавали отъезжающему. Эти палки называли *скиталами.* Когда правителям нужно было сообщить какую-нибудь важную тайну, они вырезали длинную и узкую, вроде ремня, полосу папируса, наматывали ее на свою скиталу, не оставляя на ней никакого промежутка, так чтобы вся поверхность палки была охвачена этой полосой. Затем, оставляя папирус на скитале в том виде, как он есть, писали на нем все, что нужно, а написав, снимали полосу и без палки отправляли адресату. Так как буквы на ней разбросаны в беспорядке, то прочитать написанное он мог, только взяв свою скиталу и намотав на нее без пропусков эту полосу.

Аристотелю принадлежит способ дешифрования этого шифра. Надо изготовить длинный конус и, начиная с основания, обертывать его лентой с шифрованным сообщением, постепенно сдвигая ее к вершине. В какой-то момент начнут просматриваться куски сообщения. Так можно определить диаметр скиталы.

В I в. н.э. *Ю. Цезарь* во время войны с *галлами,* переписываясь со своими друзьями в *Риме,* заменял в сообщении первую букву латинского алфавита (A) на четвертую (D), вторую (B) - на пятую (E), наконец, последнюю - на третью:

⇑ 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 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

Сообщение об одержанной им победе выглядело так:

Y H Q L Y L G L Y L F L

Император Август (I в. н. э.) в своей переписке заменял первую букву на вторую, вторую - на третью и т. д., наконец, последнюю - на первую:

⇑ 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
⇓ 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

Его любимое изречение было:

"G F T U J O B M F O U F"

Шифр *Цезаря* входит в класс шифров, называемых *"подстановка"* или *"простая замена".* Это такой шифр, в котором каждой букве алфавита соответствует буква, цифра, символ или какая-нибудь их комбинация.

1.2. Простейшие коды перестановки.

В другом классе шифров *"перестановка"* – буквы сообщения каким-нибудь способом переставляются между собой.

Одним из простейших шифров перестановки является так называемый шифр блочной перестановки. Суть его заключается в следующем. Пусть длина блока выбрана равной N. Для шифрования выбирается ключевая последовательность целых чисел от 1 до N, случайным образом «перемешанная» для N=8 примером такой последовательности будет [538461972]). Далее, текст шифруемого сообщения разделяется на блоки размера N так, чтобы одной букве текста соответствовало одно число из ключевой последовательности. Для шифрования буквы первого блока выписываются в порядке , задаваемом ключевой последовательностью (то есть первым выписывается буква, которой соответствует 1 ключевой последовательности и т.д.). После того как все буквы первого блока выписаны, аналогично выписывается второй блок и все последующие.

Для расшифровки текста первый блок зашифрованного текста выписывается в соответствии с ключевой последовательностью таким образом, чтобы первая буква зашифрованного текста оказалась на позиции, соответствующей 1 ключевой последовательности и т.д.

К классу *"перестановка"* принадлежит и шифр, называемый *"решетка Кардано"*. Для шифрования используется специальное приспособление - квадратная карточка с отверстиями, разделенная на клетки, которая при наложении на лист бумаги оставляет открытыми лишь ¼ клеток. Число строк и столбцов в карточке является четным числом. Процедура шифрования состоит в следующем. Решетка накладывается поверх бумаги и текст последовательно (слева направо, сверху вниз) вписывается в открытые клетки решетки. Далее, решетка поворачивается на 90° градусов и следующая часть текста вписывается в открытые клетки (при этом уже написанные буквы не оказываются в открытых клетках). Аналогично процедура повторяется, пока решетка не будет повернута на 360 градусов (рис. 1). После чего текст, написанный на бумаге, может быть прочитан только с помощью аналогичной решетки.

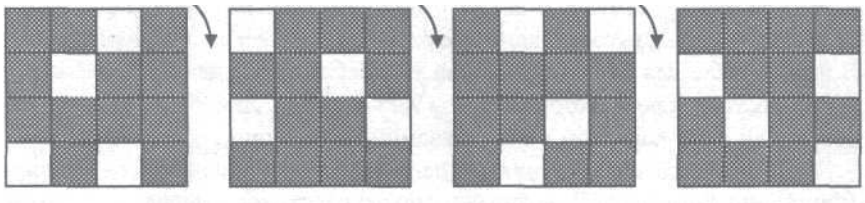


Рисунок 1. Пример использования квадратной решетки Кардано для шифрования (последовательный поворот на 90°)

1.3. Усложненные (многоалфавитные) шифры подстановки.

Для усложнения раскрываемости шифра простой подстановки цели используют *многоалфавитную систему шифрования* - систему, в которой для шифрования каждого символа употребляют тот или иной способ подстановки (алфавит подстановки) в зависимости от ключа, или от номера шифруемого символа в передаваемом сообщении.

Одним из первых способов шифрования, основанных на многоалфавитной подстановке, был так называемый шифр Вижинера. Простой вариант шифра Вижинера (*шифр Вижинера с ключевым словом*) описывается следующим образом. Для шифрования (и дешифрования) используется специальным образом составленная таблица символов *("таблица Виженера")*. При создании данной таблицы используется следующее правило: в первой строке выписывается весь алфавит, во второй строке осуществляется циклический сдвиг алфавита на одну позицию, в третьей – на две позиции и т.д.. Так получается квадратная таблица, число строк которой равно числу столбцов и равно числу букв в алфавите. На рисунке 2 представлена таблица, составленная для английского алфавита (первая строка и первый столбец выделены для удобства и не являются частью создаваемой таблицы Вижинера). Чтобы зашифровать сообщение, поступают следующим образом. Выбирается ключевое слово, известное только отправителю и получателю сообщения, (например, "монастырь") и записывается с повторением над буквами сообщения.

Чтобы зашифровать первую букву текста, необходимо выбрать столбец таблицы Вижинера, соответствующий первой букве ключевого слова, и строку, соответствующую первой букве текста. Буква, находящаяся на пересечении выделенных столбца и строки, записывается как буква зашифрованного сообщения. Аналогично шифруются все последующие буквы сообщения.

Расшифровать данное сообщение можно следующим образом. Над зашифрованным текстом сообщения, записанным в одну строку, записывается повторяющееся ключевое слово. Далее, в таблице Вижинера выбирается столбец, соответствующий первой букве ключевого слова, и в данном столбце находится буква, соответствующая первой букве зашифрованного сообщения. Строка, в которой данная буква найдена, и определяет первую букву расшифрованного текста. Аналогично процесс повторяется для всех последующих букв.

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

Рисунок 2. Пример таблицы Вижинера для английского алфавита

Кроме того, существуют более сложные варианты шифра Вижинера (*шифры Вижинера с «самоключом»*), не требующие использования ключевого слова. Шифрование в данном случае осуществляется аналогично предыдущему случаю, но в качестве буквы ключевого слова используется *предыдущая буква самого сообщения* (для шифрования первой буквы сообщения предыдущей буквой считается буква «а»).

К многоалфавитной системе шифрования можно также отнести ди*граммную подстановку*. Данный способ шифрования предполагает, что отправителю и получателю сообщения известна специальным образом созданная таблица подстановки. В данной таблице каждый столбец соответствует первой букве диграммы (двухбуквенной последовательности), а каждая строка – второй букве диграммы. В клетках таблицы вписаны случайным образом все возможные диграммы (двухбуквенные сочетания). При шифровании текст разбивается на диграммы и каждая диграмма заменяется соответствующей диграммой из таблицы.

Порядок выполнения лабораторной работы.

Общий план выполнения работы.

1. Изучить методы шифрования, представленные в данном пособии.
2. Получить от преподавателя номер варианта задания .
3. Написать программу шифрования и дешифрования текстовых файлов (конкретный метод шифрования определяется вариантом задания).
4. Сдать отчет преподавателю, ответить на контрольные вопросы, получить зачет по работе.

Контрольные вопросы.

1. Шифр простой подстановки – принцип работы.
2. Шифры перестановки –принцип работы, пример работы шифра (блочная перестановка или решетка Кардано).
3. Шифры многоалфавитной подстановки – принцип работы, пример работы шифра (шифр Вижинера, диграммная подстановка).

Требования к программам шифрования и дешифрования.

Входными данными являются текстовый файл в кодировке Windows-1251, содержащий сообщение (открытое – для программы шифрования, зашифрованное – для программы дешифрования) и текстовый файл, содержащий ключ шифрования.

В текстовом файле с сообщением могут содержаться только строчные буквы русского (или английского, на выбор) алфавита, разделенные пробелами и символами конца строки.

Выходными данными является текстовый файл в кодировке Windows-1251, содержащий выходное сообщение (открытое – для программы дешифрования, зашифрованное– для программы шифрования). Все пробелы и символы конца строки, содержавшиеся во входном файле с сообщением, должны присутствовать и в выходном файле (без их шифрования).

Варианты.

1. Реализовать программу шифрования и дешифрования для метода простой подстановки (шифр Цезаря). Ключ – число (смещение при шифровании).
2. Реализовать программу шифрования и дешифрования для метода Вижинера с ключевым словом. Ключ – слово фиксированной длины (8 символов, только русский или английский алфавит)
3. Реализовать программу шифрования и дешифрования для метода блочной перестановки. Ключ – последовательность из 18 целых чисел, разделенная пробелами.
4. Реализовать программу создания решеток Кардано заданного размера (8 на 8 ячеек). Входных данных не требуется, выходные данные – текстовый файл с записанной решеткой. В выходном файле 8 строк, разделенных символом конца строки, в каждой строке 8 символов 1 и 0 без разделения (1 – есть отверстие в решетке, 0 – нет отверстия).
5. Реализовать программу шифрования и дешифрования для решетки Кардано. Ключ – файл с записанной решеткой из варианта 3.
6. Реализовать программу шифрования и дешифрования для метода Вижинера с «самоключом» (в качестве буквы ключа используется предыдущая буква открытого текста)

7. Реализовать программу шифрования и дешифрования для метода Вижинера с «самоключом» (в качестве буквы ключа используется предыдущая буква уже зашифрованного текста)

8. Реализовать программу шифрования и дешифрования для метода диграммной подстановки. Ключ – файл с записанной таблицей диграммных подстановок, строки таблицы разделены символами конца строки, диграммы в строке разделены пробелами.

9. Реализовать программу шифрования и дешифрования для сложной блочной перестановки (для шифрования последовательно применяются две блочные перестановки с разным периодом ключа N). Ключ – текстовый файл, первой строке последовательность из 18 целых чисел, разделенных пробелами (ключ первой перестановки), во второй строке последовательность из 7 целых чисел, разделенных пробелами (ключ второй перестановки).

ЛАБОРАТОРНАЯ РАБОТА № 2

Генерирование равномерно распределенных псевдослучайных последовательностей

Цель работы

Освоить основные алгоритмы программного генерирования равномерно распределенных псевдослучайных последовательностей.

Конгруэнтные генераторы.

Линейным конгруэнтным генератором (ЛКГ) с параметрами (x_0, a, c, N) называется программный генератор РРСП, порождающий псевдослучайную последовательность $x_1, x_2, \ldots \in A$, $A = \{0,1, \ldots, N-1\}$ с помощью рекуррентного соотношения:

$$x_{t+1} = (ax_t + c) \bmod N, \; t = 0,1\ldots \qquad \textbf{(1)}$$

Параметры этого генератора (1) имеют следующий смысл: $x_0 \in A$ – начальное, или стартовое, значение; $a \in A \setminus \{0\}$ – ненулевой множитель; $c \in A$ – приращение; N – модуль, равный мощности алфавита A.

Если приращение $c = 0$, то генератор (1) называется мультипликативным конгруэнтным генератором (МКГ), а если $c \neq 0$, то смешанным конгруэнтным генератором (СКГ).

Перечислим свойства псевдослучайной последовательности, порождаемой ЛКГ:

1. Псевдослучайная последовательность (1), порождаемая ЛКГ, достигает максимального значения периода $T_{\max} = N$ тогда и только тогда, когда выполнены следующие три условия:
   a) c, N – взаимно простые, т.е. $НОД(c, N) = 1$;
   b) число $b = a - 1$ кратно p для любого простого числа $p < N$, являющегося делителем N;
   c) число b кратно *4*, если N кратно *4*.
2. Для МКГ, если x_0, N – взаимно простые, a – первообразный элемент по модулю N, а $\varphi(N)$ – максимально возможный порядок по модулю N, то псевдослучайная последовательность имеет максимальный период $T_{\max}$.

3. Для МКГ, если $N = 2^q$, $q \geq 4$, то максимально возможное значение периода $T_{\max} = 2^{q-2} = \frac{N}{4}$ псевдослучайной последовательности достигается, если $x_0 \geq 1$ – нечетно и вычет $a \bmod 8 \in \{3,5\}$.

4. «Слабость» ЛКГ и МКГ заключается в том, что если рассматривать последовательные биграммы $\left(z_1^{(t)}, z_2^{(t)}\right)$: $z_1^{(t)} = x_t$, $z_2^{(t)} = x_{t-1}$, то точки $z^t = \left(z_1^{(t)}, z_2^{(t)}\right)$, $t = 1,2\ldots$ на плоскости R^2 будут лежать на прямых из семейства $z_2 = az_1 + c - kN$, $k = 0,1\ldots$

Нелинейные конгруэнтные генераторы.

Четвертое свойство линейного и мультипликативного конгруэнтных генераторов псевдослучайных последовательностей представляет «слабость» этих генераторов и может активно использоваться для построения криптоатак в целях оценки параметров a, c, x_0. Для устранения этого недостатка используют нелинейные конгруэнтные генераторы псевдослучайных последовательностей. Наибольшее распространение получили три подхода, описание которых приводится ниже.

Квадратичные конгруэнтные генераторы.

Этот алгоритм генерации псевдослучайной последовательности $x_t \in A = \{0,1,\ldots,N-1\}$ определяется квадратичным рекуррентным соотношением:

$$x_{t+1} = \left(dx_t^2 + ax_t + c\right) \bmod N,\ t = 0,1\ldots \qquad \textbf{(2)}$$

где $x_0, a, c, d \in A$ – параметры генератора. Выбор этих параметров осуществляется на основе следующих двух свойств последовательности (2):

1. Квадратичная конгруэнтная последовательность (2) имеет наибольший период $T_{max} = N$ тогда и только тогда, когда выполнены следующие условия:

   a) c, N – взаимно простые числа;

   b) $d, a - 1$ – кратны p, где p — любой нечетный простой делитель N;

   c) d – четное число, причем

$$d = \begin{cases} (a-1) \bmod 4, & \text{если N кратно 4} \\ (a-1) \bmod 2, & \text{если N кратно 2} \end{cases}$$

   d) если N кратно 9, то либо $d \bmod 9 = 0$, либо $d \bmod 9 = 1$ и $cd \bmod 9 = 6$.

   e) Если $N = 2^q, q \geq 2$ то наибольший период $T_{max} = 2^q$ тогда и только тогда, когда c – нечетно, d – четно, a – нечетное число, удовлетворяющее соотношению: $a = (d + 1) \bmod 4$.

Генератор Эйхенауэра – Лена с обращением.

Псевдослучайная нелинейная конгруэнтная последовательность Эйхенауэра – Лена с обращением определяется следующим нелинейным рекуррентным соотношением:

$$x_{t+1} = \begin{cases} \left(ax_t^{-1} + c\right) \bmod N, & \text{если } x_t \geq 1 \\ c, & \text{если } x_t = 0 \end{cases} \qquad \textbf{(3)}$$

где x_t^{-1} – обратный к x_t элемент по модулю N, т.е. $x_t x_t^{-1} \equiv 1 (\bmod N)$; $x_0, a, c \in A$ – параметры генератора.

Выбор параметров осуществляется на основании свойства:

1. Если $N = 2^q$, a, x_0 – нечетны, c — четно, то генератор (3) имеет максимально возможный период $T_{max} = 2^{q-1}$ тогда и только тогда, когда $a \equiv 1 (\bmod 4)$, $c \equiv 2 (\bmod 4)$

Конгруэнтный генератор, использующий умножение с переносом.

При этом нелинейная конгруэнтная псевдослучайная последовательность определяется рекуррентным соотношением:

$$x_{t+1} = (ax_t + c_t) \bmod N \qquad \textbf{(4)}$$

где, в отличие от (2), «приращение» $c_t = c(x_{t-1}, x_{t-2}, \ldots, x_0)$ изменяется во времени и зависит от указанных аргументов нелинейно:

$$c_t = \left\lfloor \frac{ax_{t-1} + c_{t-1}}{N} \right\rfloor^{1} \qquad \textbf{(5)}$$

Параметрами нелинейного конгруэнтного генератора (4), (5) являются x_0, c_0, a, N.

Рекуренты в конечном поле

Обобщением мультипликативной конгруэнтной последовательности является линейная рекуррентная последовательность порядка $k \geq 1$ над конечным полем $GF(p^k)$:

$$x_{t+1} = (a_1 x_t + a_2 x_{t-1} + \ldots + a_k x_{t-k+1}) \bmod p \qquad \textbf{(6)}$$

где $a_1, \ldots, a_k \in A = \{0,1\ldots, p-1\}$ – коэффициенты рекурренты, а $x_0, \ldots, x_{-k+1} \in A$ – начальные значения рекурренты.

Параметры генератора псевдослучайной последовательности (6): $p, k, a_1, \ldots, a_{-k+1}$. Начальные значения $x_0, \ldots, x_{-k+1} \in A$ выбираются произвольно так, чтобы не обращались в ноль одновременно. Коэффициенты рекурренты $a_1, \ldots, a_k \in A$ выбираются таким образом, чтобы порождающий полином

$$f(x) = x^k - a_1 x^{k-1} - \ldots - a_{k-1}x - a_k \qquad \textbf{(7)}$$

являлся примитивным многочленом по модулю p, т.е. многочлен (7) имел корень x_*, являющийся первообразным элементом поля $GF(p^k)$. При таком

---

[1] Наибольшее целое, меньшее или равное числу в скобках.

выборе параметров достигается максимально возможный период $T_{max} = p^k - 1$ псевдослучайной последовательности (6).

Последовательности, порождаемые линейными регистрами сдвига с обратной связью.

Линейным регистром сдвига с обратной связью (Linear Feedback Shift Register, сокращенно LFSR) называется логическое устройство, схема которого изображена на рис. 1.

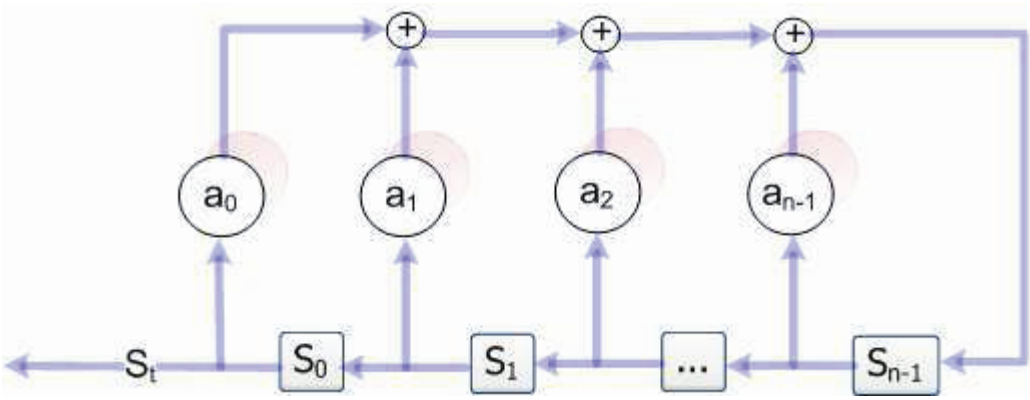


Рис. 1. Блок-схема LFSR.

LFSR состоит из n ячеек памяти, двоичные состояния которых в момент времени $t = 0,1,\ldots$ характеризуются значениями $S_0(t)$, $S_1(t)$, $\ldots$, $S_{n-1}(t) \in A = \{0,1\}$. Выходы ячеек памяти связаны не только последовательно друг с другом, но и с сумматорами $\oplus$ в соответствии с коэффициентами передачи $a_0, a_1, \ldots, a_{n-1} \in A$: если $a_i = 1$, то значение $S_i(t)$ i-ой ячейки передается на один из входов i-го сумматора; если же $a_i = 0$, то такая передача отсутствует. Полагается $a_i \equiv 0$. Состояние LFSR в текущий момент времени t задается двоичным n-вектор-столбцом $S(t) = (S_{n-1}(t), \ldots, S_0(t))'$.

Содержание ячеек LFSR с течением времени изменяется следующим образом, определяя тем самым динамику состояний LFSR:

$$S_i(t+1) = \begin{cases} S_{i+1}(t), & \text{если} \quad i \in \overline{0, n-2} \\ \sum_{j=0}^{n-1} a_j S_j(t), & \text{если} \quad i = n-1 \end{cases} \qquad \textbf{(8)}$$

Текущие значения нулевой ячейки регистра используются в качестве элементов порождаемой LFSR двоичной псевдослучайной последовательности $s_y = S_0(t)$ (см. рис. 1).

Модель (8) является частным случаем модели (7) линейной рекурренты над полем $GF(2^n)$, поэтому коэффициенты $\{a_i\}$ выбираются согласно методике, приведенной в предыдущем пункте. То есть многочлен, по которому строится LFSR, должен быть примитивным по модулю *2*. Степень многочлена является длиной сдвигового регистра. Примитивный(базовый) многочлен степени n по модулю *2* – это неприводимый многочлен, который является делителем $x^{2^n-1} - 1$, но не является делителем $x^d - 1$ для всех d, являющихся делителями $2^n - 1$. Неприводимый многочлен степени n нельзя представить в виде умножения многочленов кроме него самого и единичного.

Генераторы Фибонначи.

Общий вид рекуррентного соотношения, определяющего генератор Фибоначчи, задается уравнением

$$x_t = x_{t-r} \oplus x_{t-s}, \; t = r, r+1, r+2 \ldots \qquad \textbf{(9)}$$

где $r, s \in N(r > s)$ – параметры генератора; элемент $x_t \in V_k$ представляет собой двоичный k-вектор и действие $\oplus$ выполняется покомпонентно.

Криптостойкие генераторы на основе односторонних функций.

Для повышения стойкости алгоритмов генерации псевдослучайных последовательностей к криптоанализу в последнее время предлагается синтезировать алгоритмы на основе известных в криптографии односторонних функций. Характерное свойство односторонних (one-way) функций состоит в

том, что для вычисления значения функции по заданному значению аргумента существует полиномиально-сложный алгоритм, в то время как для вычисления аргумента по заданному значению функции полиномиально-сложного алгоритма не существует (или он не известен). Доказательство свойства односторонности функции является трудной математической задачей, поэтому в настоящее время в криптосистемах часто используются «кандидаты в односторонние функции», для которых показано лишь, что в настоящее время не известны полиномиально-сложные алгоритмы вычисления обратной функции. Примерами таких «кандидатов» являются некоторые известные криптоалгоритмы (например, DES) и хэш-функции (например, SHA-1).

Генераторы, основанные на математическом аппарате односторонних функций: ANSI X9.17, FIPS-186, Yarrow-160.

Криптостойкие генераторы, основанные на проблемах теории чисел.

Стойкость данных генераторов псевдослучайных последовательностей основывается на неразрешимости с полиномиальной сложностью (на данный момент) некоторых известных проблем теории чисел: факторизации больших чисел и дискретного логарифмирования.

Примерами, генераторов основанных на данных проблемах являются RSA-алгоритм генерации псевдослучайных последовательностей, модификация Микали-Шнорра RSA-алгоритм генерации псевдослучайных последовательностей, BBS (Blum–Blum–Shub) – алгоритм генерации псевдослучайных последовательностей.

Методы «улучшения» элементарных псевдослучайных последовательностей.

Пусть $\xi_1, \xi_2, \ldots$ – некоторая двоичная псевдослучайная последовательность, сгенерированная одним из простейших методов и называемая поэтому (в данном пункте) элементарной. Для того чтобы

построить псевдослучайную последовательность со свойствами, более близкими к свойствам РРСГТ, чем элементарная последовательность, осуществим функциональное преобразование:

$$x_1 = f_1(\xi_1, \xi_2, \ldots), \; x_2 = f_2(\xi_1, \xi_2, \ldots), \ldots$$

где $f_1(\xi_1, \xi_2, \ldots), \; f_2(\xi_1, \xi_2, \ldots), \ldots$ – некоторые функционалы, которые следует подбирать так, чтобы преобразованная последовательность $\{x_k\}$ имела вероятностное распределение, более близкое к распределению РРСП, чем распределение $\{\xi_i\}$.

Выбирая различные функционалы и метрики в пространстве вероятностных распределений, можно разработать множество методов и алгоритмов «улучшения» элементарных псевдослучайных последовательностей.

Например, алгоритм симметризации псевдослучайных последовательностей.

Комбинирование алгоритмов генерации методом Макларена – Марсальи.

Пусть имеется два простейших генератора псевдослучайных последовательностей: G_1 и G_2. Генератор G_1 порождает «элементарную» последовательность над алфавитом мощности N: $x_0, x_1, \ldots \in A(N) = \{0,1, \ldots, N-1\}$, а генератор G_2 – над алфавитом мощности K: $y_0, y_1, \ldots \in A(K) = \{0,1, \ldots, K-1\}$.

Пусть имеется вспомогательная таблица $T = \{T(0), T(1), \ldots, T(K-1)\}$, из K целых чисел (память из K ячеек).

Метод Макларена – Марсальи комбинирования последовательностей $\{x_i\}$, $\{y_i\}$ для получения выходной псевдослучайной последовательности $\{z_k\}$ состоит в следующем. Сначала T-таблица заполняется K первыми членами последовательности $\{x_i\}$. Элементы выходной последовательности

вычисляются следующим образом:

$s \leftarrow y_k,\ z_k = T(s),\ T(s) \leftarrow x_{K+k},\ k = 0,1,\ldots$

Таким образом, генератор G_2 осуществляет «случайный» выбор из T-таблицы, а также ее «случайное» заполнение «случайными» числами, порождаемыми генератором G_1.

Метод комбинирования Макларена – Марсальи позволяет ослабить зависимость между членами $\{z_k\}$ и увеличить период псевдослучайной последовательности.

Комбинирование LFSR-генераторов.

LFSR-генераторы часто используются в качестве генераторов элементарных псевдослучайных последовательностей и применяются для комбинирования генераторов. Прежде всего отметим, что LFSR-генераторы можно использовать в качестве G_1, G_2 в генераторе Макларена – Марсальи. Например, одним из способов комбинирования LFSR-генераторов является полиномиальное комбинирование элементарных последовательностей. Общая модель комбинирования LFSR-генераторов представлена на рис. 2.

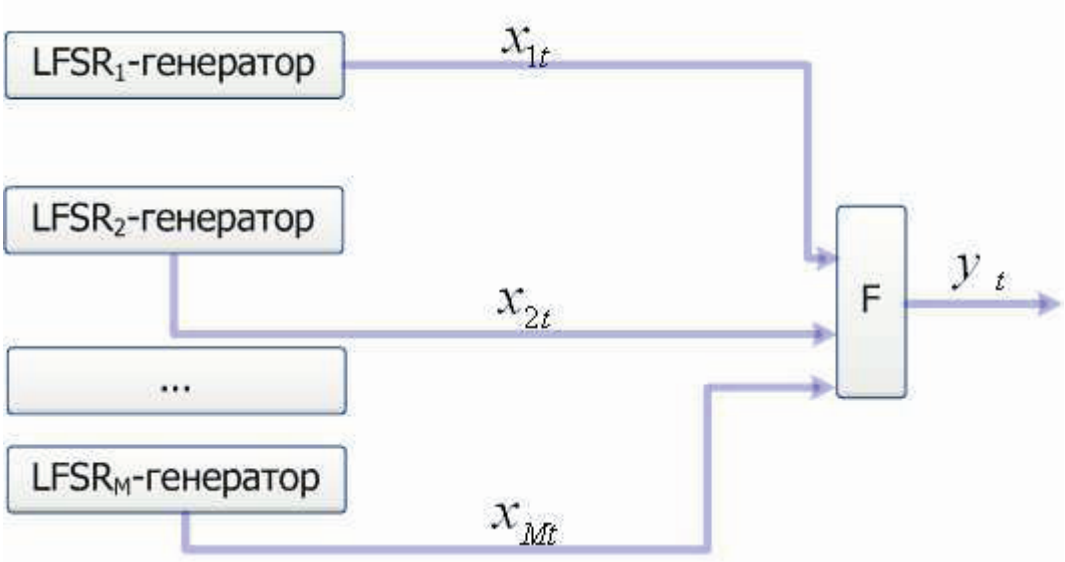


Рис. 2. Общая модель комбинирования LFSR-генераторов.

Здесь функция F общего полиномиального вида:

$$y = F(x) = \left(a_0 + \sum_{1 \le i \le M} a_i x_i + \sum_{1 \le i < j \le M} a_{ij} x_i x_j + \ldots + a_{12\ldots M} x_1 x_2 \ldots x_M \right) \bmod 2.$$

Комбинирование с помощью псевдослучайного прореживания

Рассмотрим еще один способ комбинирования двух LFSR-генераторов G_1, G_2. Пусть LFSR-генератор G_1 порождает «элементарную» двоичную последовательность $\{a_t\}$, а LFSR-генератор G_2 – двоичную «селектирующую» последовательность $\{s_t\}$. С помощью этих двух последовательностей $\{a_t\}$, $\{s_t\}$ строится выходная последовательность $\{x_t\}$, включающая те биты a_t, для которых соответствующее значение селектора $s_t = 1$; если $s_t = 0$, то значение a_t игнорируется. Такой генератор двоичной псевдослучайной последовательности называется SG-генератором.

Свойство SG-генератора выражается следующим утверждением: пусть T_a, T_s – соответственно периоды последовательностей $\{a_t\}$ и $\{s_t\}$. Если генераторы G_1, G_2 используют примитивные порождающие многочлены степеней n и m соответственно, а периоды T_a, T_s – взаимно простые числа, то выходная последовательность $\{x_t\}$ имеет период $T = (2^n - 1)2^{m-1}$.

Конгруэнтный генератор со случайными параметрами

Еще один способ комбинирования двух генераторов G_1, G_2 заключается в том, что G_2 изменяет параметры генератора G_1 с течением времени. Проиллюстрируем это в случае, когда G_2 — линейный конгруэнтный генератор:

$$x_t = (a_t x_{t-1} + b_t) \bmod N, \; t = 1, 2, \ldots \qquad \textbf{(10)}$$

где $x_0 \in A$ – некоторое стартовое значение, а $A_t = \begin{pmatrix} a_t \\ b_t \end{pmatrix} \in B, t = 1,2, \ldots$ есть некоторая псевдослучайная последовательность векторов, равномерно распределенных в B.

Доказано, что если $|B| = 3$, то наибольшее приближение распределения $\{x_t\}$ равномерному достигается, если множество параметров B имеет следующий вид: $B = \left\{ \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \end{pmatrix} \right\}$.

Задание

1. Согласно варианту реализовать приложение, генерирующие псевдослучайную равномерно распределенную последовательность произвольной длины из заданного алфавита.
2. Подобрать параметры данного генератора таким образом, чтобы период последовательности имел максимальное значение.

Вариант	Генератор	Размер алфавита	Дополнение
1	Комбинирование алгоритмов генерации методом Макларена – Марсальи	N=10 K=5	
2	Комбинирование с помощью псевдослучайного		*Рассмотреть следующие случаи*

	прореживания		*генераторов G_1, основанных на примитивных многочленах:* 1. $f(x) = x^7 + x^3 + x^2 + x + 1$ 2. $f(x) = x^7 + x + 1$ 3. $f(x) = x^7 + x^3 + 1$ *Какие из этих многочленов примитивны по модулю 2?*
3	Генератор Эйхенауэра – Лена с обращением	N=20	
4	LFSR		*Рассмотреть следующие случаи генераторов, основанных на примитивных многочленах:* 1. $f(x) = x^7 + x + 1$ 2. $f(x) = x^7 + x^5 + x^3 + 1$ 3. $f(x) = x^7 + x^6 + x^5 + x^2 + 1$ *Какие из этих многочленов примитивны по модулю 2?*
5	Конгруэнтный генератор со случайными параметрами	N=15	*Рассмотреть следующие случаи:* 1. $B = \left\{\begin{pmatrix}3\\0\end{pmatrix}, \begin{pmatrix}3\\1\end{pmatrix}, \begin{pmatrix}3\\2\end{pmatrix}\right\}$ 2. B выбрать произвольно, $\lvert B\rvert = 5$
6	Линейный конгруэнтный генератор	N=15	*Рассмотреть следующие случаи:* 1. Мультипликативный генератор Лемера: $c = 0$ 2. Смешанный генератор
7	LFSR		*Рассмотреть следующие случаи генераторов, основанных на примитивных многочленах:* 1. $f(x) = x^6 + x^3 + 1$ 2. $f(x) = x^6 + x^2 + x + 1$ 3. $f(x) = x^6 + x + 1$ *Какие из этих многочленов примитивны по модулю 2?*
8	Аддитивный генератор	N=30	$x_{t+1} = (x_t + x_{t-1}) \bmod N$
9	Генераторы Фибонначи	N=2	
10	Комбинирование с помощью псевдослучайного прореживания		*Рассмотреть следующие случаи генераторов G_1, основанных на примитивных многочленах:* 1. $f(x) = x^5 + x^2 + 1$ 2. $f(x) = x^5 + x^4 + x^3 + x^2 + 1$ 3. $f(x) = x^5 + 1$ *Какие из этих многочленов примитивны по модулю 2?*
11	Квадратичный конгруэнтный	N=10	*Рассмотреть следующие случаи:* 1. Генератор Ковью: $c = 0$

	генератор		2. $c \neq 0$
12	LFSR		*Рассмотреть следующие случаи генераторов, основанных на примитивных многочленах:* 1. $f(x) = x^5 + x^2 + 1$ 2. $f(x) = x^5 + 1$ 3. $f(x) = x^5 + x^4 + x^2 + 1$ *Какие из этих многочленов примитивны по модулю 2?*
13	Конгруэнтный генератор, использующий умножение с переносом.	N=30	
14	LFSR		*Рассмотреть следующие случаи генераторов, основанных на примитивных многочленах:* 1. $f(x) = x^4 + x$ 2. $f(x) = x^4 + x^3 + x^2 + x + 1$ 3. $f(x) = x^4 + x^1 + 1$ *Какие из этих многочленов примитивны по модулю 2?*
15	Генератор Эйхенауэра – Лена с обращением	N=15	

Контрольные вопросы

1. С какой целью в криптологии используют псевдослучайные последовательности?
2. Зачем генерировать псевдослучайные последовательности с максимальным периодом?
3. Равномерно распределенная случайная последовательность.
4. Классификация алгоритмов генерации псевдослучайных последовательностей.
5. Конгруэнтные генераторы.
6. Рекуренты в конечном поле.
7. Криптостойкие генераторы на основе односторонних функций.
8. Криптостойкие генераторы, основанные на проблемах теории чисел.
9. Методы «улучшения» элементарных псевдослучайных последовательностей.

ЛАБОРАТОРНАЯ РАБОТА №3

Тестирование чисел на простоту и построение больших простых чисел

Цель работы

Освоить основные программные методы тестирования чисел на простоту

3.1. Метод пробных делений

Если n – составное, то $n = a \cdot b$, где $1 < a \le b$ причем $a \le \sqrt{n}$. Поэтому для $d = \overline{2, \lfloor \sqrt{n} \rfloor}$ мы проверяем, делится ли n на d? Если делитель числа n не будет найден, то n – простое. В противном случае будет найден минимальный простой делитель числа n, т.е. n будет разложено на два множителя. Сложность метода составляет $O(\sqrt{n})$ арифметических операций с целыми числами.

Возможны модификации этого метода. Например, мы можем проверить, делится ли n на *2* и на *3* , и если нет, то перебираем далее только числа d вида $1 + 6j, 5 + 6j,\ j = 1, 2, \ldots$.

3.2. Решето Эратосфена

Если мы хотим составить таблицу всех простых чисел среди чисел $2, 3, \ldots, n$ то нужно сначала вычеркнуть все числа, делящиеся на *2*, кроме *2*. Затем взять число *3* и вычеркнуть все последующие числа, делящиеся на *3* . Затем взять следующее не вычеркнутое число (т. е. *5*) и вычеркнуть все

последующие делящиеся на него числа, и так далее. В итоге останутся лишь простые числа.

Для реализации метода нужен большой объем памяти ЭВМ, однако для составления таблиц простых чисел он является наилучшим.

Всего требуется $O(\log^3 n)$ арифметических операций.

3.3. Критерий Вильсона

Теорема 1[1].

Для любого n следующие условия эквивалентны:

1. n – простое;
2. $(n-1)! \equiv -1 (\mathrm{mod}\ n)$

Данный критерий иногда бывает удобен в доказательствах, но применять его для проверки простоты невозможно ввиду большой трудоемкости.

3.4. Тест на основе малой теоремы Ферма

Малая теорема Ферма утверждает, что если n – простое, то выполняется условие: $\forall a = \overline{2, n-1}$ имеет место сравнение:

$$a^{n-1} \equiv 1 (\mathrm{mod}\ n) \qquad \textbf{(1)}$$

Обратное утверждение неверно.

Из этой теоремы следует, что если (1) не выполнилось хотя бы для одного числа a в интервале $[2; n-1]$, то n – составное. Поэтому можно предложить следующий вероятностный тест простоты:

1. Выбираем случайное число из интервала $[1; n-1]$ и проверяем с помощью алгоритма Евклида[2] условие $НОД(a, n) = 1$.
2. Проверяем выполнимость сравнения $a^{n-1} \equiv 1 (\mathrm{mod}\ n)$.
3. Если сравнение (1) не выполнено, то ответ n – составное

---

[1] В 1770 г. Э. Варинг опубликовал эту теорему, приписываемую Д. Вильсону.

[2] См. прил. 1.

4. Если сравнение (1) выполнено то ответ неизвестен но можно повторить тест еще раз.

Если выполняется сравнение (1), то говорят, что число n является псевдопростым по основанию a. Заметим, что существует бесконечно много пар чисел (a, n), где n – составное и псевдопростое по основанию a. Например, при $(a, n) = (2, 341)$ получаем $2^{340} = \left(2^{10}\right)^{34} \equiv 1 (\mathrm{mod}\ 341)$, хотя $341 = 11 \cdot 31$.

Особый случай составляют составные числа, для которых условие сравнения выполняется при всех основаниях. Они называются псевдопростыми числами, или числами Кармайкла.

Таким образом, при применении описанного выше теста может возникнуть три ситуации:

- число n простое и тест всегда говорит «не известно»;
- число n составное и не является числом Кармайкла; тогда с вероятностью успеха не меньше $\frac{1}{2}$ тест дает ответ «n – составное»;
- число n составное и является числом Кармайкла, тогда тест всегда дает ответ «не известно».

Числа Кармайкла являются достаточно редкими. Так, имеется всего *2163* чисел Кармайкла не превосходящих $25 \cdot 10^9$. До 10^5 числами Кармайкла являются только следующие *16* чисел: *561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973* и *75361*. Проверка того, является ли заданное число числом Кармайкла, требует нахождения разложения числа на простые сомножители, т. е. факторизации числа. Поскольку задача факторизации чисел является более сложной, чем задача проверки простоты, то предварительная отбраковка чисел Кармайкла не представляется возможной. Поэтому в приведенном выше тесте простые числа и числа Кармайкла полностью неразличимы.

6.5. Тест Соловея – Штрассена

Теорема 2.

Для любого нечетного n следующие условия эквивалентны:

1. n – простое;
2. $\forall a$:

$$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (\mathrm{mod}\ n) \qquad \textbf{(2)}$$

где $\left(\frac{a}{n}\right)$ – символ Лежандра.

Р. Соловей и В.Штрассен предложили следующий вероятностный тест для проверки простоты чисел:

1. Выбираем случайное число a из интервала $[1;\ n-1]$ и проверяем с помощью алгоритма Евклида условие $НОД(a, n) = 1$.
2. Если оно не выполняется, то n – составное.
3. Проверяем выполнимость сравнения (2).
4. Если оно не выполняется, то n – составное.
5. Если сравнение выполнено, то ответ неизвестен (и тест можно повторить еще раз).

Сложность данного теста, как и теста на основе малой теоремы Ферма, оценивается величиной $O(\log^3 n)$.

Данный тест полностью аналогичен тесту на основе малой теоремы Ферма, однако, он обладает решающим преимуществом – при его использовании возникает только две ситуации:

- число n простое и тест всегда говорит «не известно»;
- число n составное и тест с вероятностью успеха не меньше $\frac{1}{2}$ дает ответ «n – составное».

После повторения теста k раз вероятность неотбраковки составного числа не превосходит $\frac{1}{2^k}$.

3.6. Тест Леманна.

Далее приведена последовательность действий при проверке простоты числа n:

1. Выбрать случайно число a, меньшее n.
2. Вычислить $t = a^{\frac{p-1}{2}} \bmod n$.
3. Если $t \neq \pm 1$ (или или), то n не является простым.
4. Если $t \equiv \pm 1$ (или или), то вероятность того, что число n не является простым, не больше $\frac{1}{2}$.

Повторите эту проверку k раз. Если результат вычислений равен 1 или – 1, но не всегда равен 1, то n является простым числом с вероятностью ошибки $\frac{1}{2^k}$.

3.7. Тест Рабина – Миллера[4]

Пусть n – нечетное и $n - 1 = 2^s \cdot t$, t – нечетное. Если число n является простым, то при всех $a \geq 2$ выполняется сравнение $a^{n-1} \equiv 1 (\bmod n)$. Поэтому, рассматривая элементы $a^t, a^{2t}, \ldots, a^{2^{s-1}t}$ можно заметить, что либо среди них найдется равный $-1 (\bmod n)$, либо $a^t \equiv 1 (\bmod n)$.

На этом замечании основан следующий вероятностный тест простоты:

1. Выбираем случайное число a из интервала $[1; n - 1]$ и проверяем с помощью алгоритма Евклида условие $НОД(a, n) = 1$.
2. Если оно не выполняется, то ответ «n – составное».
3. Вычисляем $a^t \bmod n$.

---

4 Предложен М. Рабином в 1980 г.

4. Если $a^t \equiv \pm 1 (\mathrm{mod}\ n)$, то переходим к шагу 1.

5. Вычисляем $a^t \bmod n, a^{2t} \bmod n, \ldots, a^{2^{s-1}t} \bmod n$ до тех пор, пока не появится -1.

6. Если ни одно из этих чисел не равно -1, то ответ «n – составное»;

7. Если мы достигли n, то ответ неизвестен (и тест можно повторить еще раз).

Арифметическая сложность данного теста, очевидно, составляет $O(s \cdot n)$.

После повторения данного теста k раз вероятность неотбраковки составного числа не превосходит $\frac{1}{4^k}$.

Тест Рабина – Миллера всегда сильнее теста Соловея – Штрассена. Точнее, если при фиксированном n число a проходит тест Рабина – Миллера и не показывает, что n составное, то оно проходит тест Соловея – Штрассена с тем же результатом.

3.8. Полиномиальный тест распознавания простоты.

Данный алгоритм основан на следующем критерии простоты.

Теорема 3.

Пусть числа a, n взаимно просты. Тогда n – простое в том и только в том случае, когда выполнено сравнение:

$$(x - a)^n \equiv (x^n - a) (\mathrm{mod}\ n) \qquad \textbf{(3)}$$

Приведем сам алгоритм.

Вход: целое $n > 1$.

1. Если число n имеет вид a^b, то ответ «n – составное».
2. $r \leftarrow 2$
3. Цикл пока $r < n$:
4. Если $НОД(r, n) \neq 1$, то ответ «n – составное».

5. Если r – простое, то вычислить q – наибольший простой делитель $r - 1$; если $\left(q > 4\sqrt{r}\ \log_2 n\right)$ и $\left(n^{\frac{r-1}{q}} \not\equiv 1 \pmod r\right)$, то выйти из цикла.
6. $r \leftarrow r + 1$.
7. Завершить цикл.
8. Если $r = n$, то ответ «n – составное».
9. Если $n - 1 \le \left[2\sqrt{r}\ \log_2 n\right]$, то $\forall a = \overline{r, n-1}$ проверить выполнение условия: $НОД(a, n) = 1$.
10. Если $n - 1 > \left[2\sqrt{r}\ \log_2 n\right]$, то $a = \overline{1, 2\sqrt{r}\ \log_2 n}$ проверить выполнение условия: $(x - a)^n \equiv \left(x^n - a\right)\left(\mathrm{mod}\ x^r - 1\right)$.
11. Если на шаге 9-10 нарушились равенства, то ответ «n – составное».
12. Если мы дошли до этого шага, то ответ «n – простое».

Задание

1. Реализовать приложение, позволяющее генерировать простое число по следующей схеме (с использованием, например, теста Рабина – Миллера):

    1. Сгенерируйте случайное p-битовое число n.
    2. Установите старший и младший биты равными *1*. (Старший бит гарантирует требуемую длину простого числа, а младший бит обеспечивает его нечетность.)
    3. Убедитесь, что n не делится на небольшие простые числа: 3, 5, 7, 11, и т.д. Во многих реализациях проверяется делимость n на все простые числа, меньшие 256. Наиболее эффективной является проверка на делимость для всех простых чисел, меньших 2000.

    Все простые числа, не превосходящие *256* перечислены ниже:
    2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,
    83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,
    163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,
    241,251.

4. Выполните один тест Рабина – Миллера проверки простоты числа для некоторого случайного a. Если n проходит тест, сгенерируйте другое случайное a и повторите проверку. Выбирайте небольшие значения a для ускорения вычислений. Выполните пять тестов. (Одного может показаться достаточным, но выполните пять.) Если n не проходит одной из проверок, сгенерируйте другое n и попробуйте снова.

Можно не генерировать n случайным образом каждый раз, но последовательно перебирать числа, начиная со случайно выбранного до тех пор, пока не будет найдено простое число.

Вместо теста Рабина – Миллера следует использовать тест, заданный в варианте.

Вариант	Алгоритм
1	Тест Соловея – Штрассена
2	Тест Леманна.
3	Критерий Вильсона+Решето Эратосфена
4	Тест Рабина – Миллера
5	Тест на основе малой теоремы Ферма
6	Полиномиальный тест распознавания простоты.

Контрольные вопросы

1. Простые числа.
2. Метод пробных делений
3. Решето Эратосфена
4. Критерий Вильсона
5. Тест на основе малой теоремы Ферма

Приложение 1

Код функции, реализующий алгоритм Евклида

```
/* возвращает НОД (gcd) x и y */
int gcd (int x, int y) {
int g;
if (x < 0)
x = -x;
if (У < 0)
y = -y;
if (x + y == 0 )
ERROR ;
g = y;
while   (x > 0)
{
g = x; x = y % x;
y = g;
}
```

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Кафедра геоинформатики и информационной безопасности

КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Тест по курсу лекций

Самара 2013

1. Ошибка пропуска бита в передаваемом шифротексте приведет к невозможности расшифровать все последующие биты сообщения в случае а) блочного шифра б) **синхронного поточного шифра** в) самосинхронизирующегося поточного шифра.
2. Нахождение коллизии хэш-функции это а) подбор двух произвольных сообщений с одинаковым значением хэша б) подбор произвольного сообщения с заданным значением хэша в) **оба предыдущих варианта**
3. Стойкость алгоритм RSA к вычислению секретного ключа по открытому ключу основана на а) экспоненциальной сложности задачи об укладке рюкзака б) **экспоненциальной сложности задач факторизации и вычисления дискретного логарифма** в) полиномиальной сложности возведения числа в большую степень по модулю и экспоненциальной сложности решения задачи об укладке рюкзака
4. Для проведения успешной атаки «человек посередине» на протокол Диффи-Хеллмана обязательно наличие а) **активного нарушителя** б) пассивного нарушителя в) обоих типов нарушителей
5. Период бинарного РСЛОС (регистр сдвига с линейной обратной связью) при использовании примитивного многочлена обратной связи равен (n – число битовых ячеек памяти в регистре) а) 2^n; б) 2^{n-1}; в) $\mathbf{2^n-1}$; г)2^{n-1} -1
6. Практическая невозможность восстановления секретного общего ключа в протоколе Диффи-Хеллмана обоснована вычислительной сложностью а) задачи факторизации б)**вычисления дискретного логарифма** в) возведения простого числа в степень по модулю.
7. Рассеяние и смешивание, согласно рекомендациям К. Шеннона, должны быть свойствами а) **шифра** б) алгоритмов электронной цифровой подписи в) криптографически стойкого генератора псевдослучайных чисел
8. Алгоритм RSA применяется для а) шифрования б) электронной цифровой подписи в) **обоих предыдущих вариантов**
9. Протокол Диффи-Хеллмана позволяет : а) шифровать сообщение открытым ключом ; б) **согласовать двум или более участникам секретный ключ по открытому каналу связи ;** в) согласовать двум или более участникам секретный ключ по закрытому (защищенному от перехвата и подмены сообщений) каналу связи
10. Подстановочный узел (S-Box) является традиционным элементом а) генератора гаммы в поточном шифре б) **сети Фейстеля** в) алгоритмов быстрого возведения простого числа в целую степень по модулю
11. Схема аутентификации Лэмпарда предполагает а) **многократное вычисление хэш-функции от секретного ключа для генерации одноразовых ключей аутентификации** б) многократном вычислении имитоприставки от текущего системного времени в текстовом формате в) использование слепой электронной цифровой подписи
12. Алгоритм формирования имитоприставки (MAC) отличается от алгоритма HMAC следующим: а) HMAC является сертифицированным в РФ алгоритмом электронной цифровой подписи б) MAC является частным случаем алгоритма HMAC при использовании ключа фиксированной длины в) **HMAC является частным случаем MAC и предполагает вычислении только хэш-функции при формировании имитоприставки**
13. Алгоритм Рабина-Миллера это: а) детерминированный тест на простоту числа; б) **вероятностный тест на простоту числа**; в) алгоритм генерации псевдослучайных чисел.
14. Одноразовый блокнот был предложен в качестве: а) **модели идеального шифра** б) протокола взаимной аутентификации по открытому каналу связи; в) алгоритма вычисления хэш-функции

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

СОГЛАСОВАНО

Управление образовательных программ

_____/ А.В. Дорошин /

"_____"_____ 20___ г.

УТВЕРЖДАЮ

Проректор по учебной работе

_____/ Ф.В. Гречников /

"____" _____ 20___ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Наименование модуля (дисциплины)

Криптографические методы защиты информации

Цикл, в рамках которого происходит освоение модуля (дисциплины)	Профессиональный цикл
Часть цикла	С3.Б.13

Код учебного плана

090303.2.65-2011-О-П-5г00м

Факультет	информатики
Кафедра	геоинформатики и информационной безопасности
Курс	4
Семестр	8
Лекции (СЛ)	54
Семинарские и практические занятия (СП)	0
Лабораторные занятия (СЛР)	36
Контроль самостоятельной работы / Индивидуальные занятия (КСР / ИЗ)	0
Самостоятельная работа (СРС)	90
Всего (Всего с экзаменами)	180

Экзамен	8
Зачет	-

Согласовано: ФЛА

Наименование стандарта, на основании которого составлена рабочая программа:

Федеральный государственный образовательный стандарт высшего профессионального образования по направлению подготовки 090303 «Информационная безопасность автоматизированных систем»

Соответствие содержания рабочей программы, условий ее реализации, материально-технической и учебно-методической обеспеченности учебного процесса по дисциплине всем требованиям государственных стандартов подтверждаем.

Составители:

д.ф.-м.н., проф. Чернов В.М. _____
(подпись)

Заведующий кафедрой:

д.т.н., проф. Сергеев В.В. _____
(подпись)

Рабочая программа обсуждена на заседании кафедры

геоинформатики и информационной безопасности

Протокол №___ от "____" _____ 20___ г.

Наличие основной литературы в фондах научно-технической библиотеки (НТБ) подтверждаем:

Директор НТБ _____ /_____/
(подпись) (расшифровка подписи)

Согласовано:

Декан _____ /_____/
(подпись) (расшифровка подписи)

1 Цели и задачи модуля (дисциплины), требования к уровню освоения содержания

1.1 Перечень развиваемых компетенций

- ОК-2 Способность иметь представление о современном состоянии и проблемах прикладной математики и информатики, истории и методологии их развития;
- ОК-3 Способность использовать углубленные теоретические и практические знания в области прикладной математики и информатики;
- ОК-4 Способность самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе, в новых областях знаний, непосредственно не связанных со сферой деятельности, расширять и углублять свое научное мировоззрение;
- ОК-5 Способность порождать новые идеи и демонстрировать навыки самостоятельной научно-исследовательской работы и работы в научном коллективе;
- ПК-1 Способность проводить научные исследования и получать новые научные и прикладные результаты;
- ПК-3 Способность углубленного анализа проблем, потановки и обоснования задач научной и проектно-технологической деятельности;
- ПК-12 Способность участвовать в деятельности профессиональных сетевых сообществ по конкретным направлениям.

1.2 Цели и задачи изучения модуля (дисциплины)

1 Наделение студентов знаниями основных принципов криптографической защиты информации, математических моделей и методов исследования криптографических систем, основных принципов проектирования систем криптографической защиты информации.
2 Выработка у студентов приёмов и навыков решения разнообразных задач, связанных с проектированием, анализом и программной реализацией систем криптографической защиты информации.
3 Ознакомление студентов с современными технологическими стандартами в области разработки систем криптографической защиты информации.

1.3 Требования к уровню подготовки студента, завершившего изучение данного модуля (дисциплины)

Студенты, завершившие изучение данной дисциплины, должны
знать: основные термины и понятия, используемые при описании криптографических средств защиты информации, основные задачи, решаемые посредством криптографических средств защиты информации, методы классификации алгоритмов шифрования, основные математические методы анализа и синтеза криптографических систем, криптографические методы решения задач аутентификации и обеспечения целостности данных.
уметь: применять математические методы анализа криптографических систем для оценки стойкости криптографических систем, проектировать криптографические системы для решения типовых задач защиты информации (обеспечение конфиденциальности, целостности, имитостойкости).

1.4 Связь с предшествующими модулями (дисциплинами)

Курс базируется на знаниях студентов, полученных при изучении курсов:
1 Информатика (И).
2 Теория информации (ТИ).
3 Компьютерная алгебра (КА).

1.5 Связь с последующими модулями (дисциплинами)

Знания, полученные студентами в рамках настоящего курса, используются при выполнении ими преддипломной практики и дипломного проектирования.

2 Содержание рабочей программы (модуля)

Семестр 1		
СЛ 0,3 54 часов 1,5 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 1	Основные понятия и задачи криптографии: основной объект криптографии, криптография и криптоанализ, история развития и формирования методов; шифрование и дешифрование, понятия открытого текста, шифротекста, ключа шифрования, основные проблемы шифрования,
		Классы шифров: - шифры перестановки, шифры замены, шифры многоалфавитной замены, шифр Вижинера; криптоанализ шифров перестановок и многоалфавитной замены; модель открытого текста на основе частотных характеристик языка; энтропия источника и избыточность
		Криптографическая стойкость шифров: классификация угроз и атак, модель открытого канала связи, пассивный и активный нарушитель, теоретико-информационный подход к оценке криптографической стойкости; требования к совершенному шифру по Шеннону. Шифры, стойк

		Современные симметричные шифры: блочные шифры, сеть Фейстеля; алгоритм шифрования ГОСТ 28147-89, режимы использования блочных шифров; блочные шифры, основанные на операциях в поле GF(2n), алгоритм шифрования AES; существующие атаки на алгоритмы AES и ГОСТ
		Шифры с открытым ключом: понятие асимметричной (двухключевой) криптосистемы, схема обмена открытыми ключами, гибридная криптосистема с сеансовым ключом; односторонняя функция, односторонняя функция с секретом, понятие криптографически надежной односторон
		Задачи аутентификации и контроля целостности. криптографические хеш-функции: понятие хеш-функции, типы хеш-функций, требования к криптографическим хеш-функциям, коллизии 1 и 2 рода, «парадокс дней рождения»; односторонняя функция со сжатием, построение хе
		Электронная цифровая подпись: понятие электронной цифровой подписи (ЭЦП), требования к ЭЦП; использование алгоритмов RSA и Эль-Гамаля в режиме формирования ЭЦП; использование хеш-функции для повышения надежности ЭЦП.
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 1	
	Традиционные 0	
СЛР 0,2 36 часов 1 ЗЕТ	Активные 1	
	Интерактивные 0	Примеры простейших шифров, шифр Цезаря, шифр Кардано.. Криптоанализ шифров простой и многоалфавитной замены с использованием частотных характеристик языка

		Расчет энтропии источника открытых текстов. Расчет расстояния единственности для текстов на естественном языке. Расчет мощности пространства ключей и вычислительной сложности атаки прямым перебором
		Вычисление периода псевдослучайных последовательностей, вычисление примитивных многочленов в поле GF(2^n)
		Построение асимметричной системы на основе задачи об укладке рюкзака. Оценка вычислительной сложности атаки прямого перебора для задач дискретного логарифмирования и факторизации
	Традиционные 0	
КСР 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СРС 0,5 90 часов 2,5 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 1	Поиск и изучение дополнительной литературы и современных статей по темам изученного материала, включая электронные издания
		Подготовка к лабораторным занятиям

3 Инновационные методы обучения

Использование современных программных и технических средств представления материала (презентации PowerPoint, проектор) для изложения материалов лекций; Выполнение лабораторных работ в специализированных классах.

4 Технические средства и материальное обеспечение учебного процесса

Проектор, предназначенный для демонстрации материалов лекционных занятий. Учебные классы, оборудованные спецтехникой.

5 Учебно-методическое обеспечение

5.1 Основная литература

1 Смарт Н. Криптография / Н. Смарт; пер. с англ. С.А. Кулешова; под. ред. С.К. Ландо. – М.: Техносфера, 2005. (20 экземпляров)
2 Рожков, Александр Викторович. Теоретико-числовые методы в криптографии: учеб. пособие/ А.В. Рожков, О.В. Ниссенбаум – Тюмень,Тюм. гос ун-т, 2007. (14 экземпляров)
3 Корт, Семен Станиславович. Теоретические основы защиты информации : [учеб пособие по группе специальностей в обл. информ. безопасности] / С.С. Корт – М. : Гелиос АРВ, 2004 (45 экземпляров)

5.2 Дополнительная литература

1 Баричев, Сергей Геннадьевич. Основы современной криптографии: учебный курс:[для вузов] / С. Г. Баричев, В.В Гончаров, Р.Е. Серов – 2-е изд., перераб. и доп. – М. : Горячая линия – Телеком, 2002. (15 экз.)
2 Лапонина, Ольга Робертовна. Основы сетевой безопасности: криптографические алгоритмы и протоколы взаимодействия : курс лекций: учебное пособие [по специальности 510200 «Прикладная математика и информатика»] / Лапонина О.Р. ; под ред. В.А. Сухомлина ; Интернет - ун-т информ. технологий. – М.: ИНТУИТ. Ру, 2005 (2 экз.)
3 Молдовян Н. А. Криптография : от примитивов к синтезу алгоритмов / Н.А. Молдовян, А.А. Молдовян, М.А. Еремеев. – СПб. : БХВ-Петербург, 2004. (5 экз.)

5.3 Электронные источники и интернет ресурсы

http://www.intuit.ru/catalog/database

5.4 Методические указания и рекомендации

Текущий контроль знаний студентов в семестре завершается на отчетном занятии, результатом которого является допуск или недопуск студента к экзамену по дисциплине. Основанием для допуска к экзамену является выполнение студентом выполнения всех лабораторных
Экзамен проводится согласно положению о текущем и промежуточном контроле знаний студентов, утвержденном ректором института. Экзаменационная оценка ставится на основании письменного и устного ответов студента по экзаменационному билету, а также, при необхо