**Оглавление**

Введение

Глава I. Криптология

1.1 Шифрование – метод защиты информации

1.2 История развития криптологии

1.3 Криптология в наши дни

1.4 Классификация алгоритмов шифрования

Глава II. Рассмотрение алгоритмов

2.1 Симметричные алгоритмы шифрования

2.1.1 Потоковые шифры

2.1.2 Блочные шифры

2.2 Ассиметричные алгоритмы шифрования

2.2.1 Алгоритм Диффи-Хелмана

2.2.2 Агоритм RSA

2.2.3 ElGamal

2.3 Сравнение симметричных и ассиметричных алгоритмов шифрования

2.4 Использование инструментов криптографии в Delphi-приложениях

2.4.1 CryptoAPI

2.4.2 Подключение к криптопровайдеру. Контейнеры ключей

2.4.3 Шифрование на основе пользовательских данных или пароля

2.5 Постановка задачи

2.6 Реализация задачи

2.6.1 Краткая характеристика среды Delphi7

2.6.2 Алгоритм решения задачи

2.6.3 Таблица сообщений

Заключение

Литература

Введение

Многим из нас приходилось видеть личную информацию в различных свободно продаваемых базах данных. Большинство людей это справедливо возмущало, и они требовали прекратить свободное распространение личных данных. Но проблема заключалась в том, что личные данные не были четко классифицированы, и не ясно было, что можно распространять, а что нельзя.

Точки над i наконец расставили в законе Российской Федерации №152-ФЗ от 27 июля 2006 года. И теперь мы знаем, что персональные данные это любая информация, относящаяся к определенному или определяемому на основании такой информации физическому лицу (субъекту персональных данных), в том числе его фамилия, имя, отчество, год, месяц, дата и место рождения, адрес, семейное, социальное, имущественное положение, образование, профессия, доходы, другая информация.

Но закон накладывает определенные обязательства на операторов персональных данных – те компании и организации, которые оперируют такими данными. Согласно закону, с 1 января 2010 года все операторы персональных данных должны обеспечить защиту персональных данных в соответствии с требованиями закона 152-ФЗ. А «лица, виновные в нарушении требований настоящего Федерального закона, несут гражданскую, уголовную, административную, дисциплинарную и иную предусмотренную законодательством Российской Федерации ответственность».

В своей работе я хочу рассмотреть алгоритмы, при помощи которых мы можем защитить свои персональные данные.

Глава I. Криптология

**1.1 Шифрование - метод защиты информации**

Испокон веков не было ценности большей, чем информация. ХХ век - век информатики и информатизации. Технология дает возможность передавать и хранить все большие объемы информации. Это благо имеет и оборотную сторону. Информация становится все более уязвимой по разным причинам:

* возрастающие объемы хранимых и передаваемых данных;
* расширение круга пользователей, имеющих доступ к ресурсам ЭВМ, программам и данным;
* усложнение режимов эксплуатации вычислительных систем.

Поэтому все большую важность приобретает проблема защиты информации от несанкционированного доступа (НСД) при передаче и хранении. Сущность этой проблемы - постоянная борьба специалистов по защите информации со своими "оппонентами".

Защита информации - совокупность мероприятий, методов и средств, обеспечивающих:

* исключение НСД к ресурсам ЭВМ, программам и данным;
* проверку целостности информации;
* исключение несанкционированного использования программ (защита программ от копирования).

Очевидная тенденция к переходу на цифровые методы передачи и хранения информации позволяет применять унифицированные методы и алгоритмы для защиты дискретной (текст, факс, телекс) и непрерывной (речь) информации.

Испытанный метод защиты информации от НСД - шифрование (криптография). Шифрованием (encryption) называют процесс преобразования открытых данных (plaintext) в зашифрованные (шифртекст, ciphertext) или зашифрованных данных в открытые по определенным правилам с применением ключей. В англоязычной литературе зашифрование/расшифрование - enciphering/deciphering.

С помощью криптографических методов возможно:

* шифрование информации;
* реализация электронной подписи;
* распределение ключей шифрования;
* защита от случайного или умышленного изменения информации.

К алгоритмам шифрования предъявляются определенные требования:

* высокий уровень защиты данных против дешифрования и возможной модификации;
* защищенность информации должна основываться только на знании ключа и не зависеть от того, известен алгоритм или нет (правило Киркхоффа);
* малое изменение исходного текста или ключа должно приводить к значительному изменению шифрованного текста (эффект "обвала");
* область значений ключа должна исключать возможность дешифрования данных путем перебора значений ключа;
* экономичность реализации алгоритма при достаточном быстродействии;
* стоимость дешифрования данных без знания ключа должна превышать стоимость данных.

**1.2 История развития криптологии**

Криптология - древняя наука и обычно это подчеркивают рассказом о Юлии Цезаре (100 - 44 гг. до н. э.), переписка которого с Цицероном (106 - 43 гг. до н. э.) и другими "абонентами" в Древнем Риме шифровалась. Шифр Цезаря, иначе шифр циклических подстановок, состоит в замене каждой буквы в сообщении буквой алфавита, отстоящей от нее на фиксированное число букв. Алфавит считается циклическим, то есть после Z следует A. Цезарь заменял букву буквой, отстоящей от исходной на три.

Сегодня в криптологии принято оперировать символами не в виде букв, а в виде чисел, им соответствующих. Так, в латинском алфавите можем использовать числа от 0 (соответствующего A) до 25 (Z). Обозначая число, соответствующее исходному символу, x, а закодированному - y, можем записать правило применения подстановочного шифра:

y = x + z (mod N), (1)

где z - секретный ключ, N - количество символов в алфавите, а сложение по модулю N - операция, аналогичная обычному сложению, с тем лишь отличием, что если обычное суммирование дает результат, больший или равный N, то значением суммы считается остаток от деления его на N.

Шифр Цезаря в принятых обозначениях соответствует значению секретного ключа z = 3 (а у Цезаря Августа z = 4). Такие шифры раскрываются чрезвычайно просто даже без знания значения ключа: достаточно знать лишь алгоритм шифрования, а ключ можно подобрать простым перебором (так называемой силовой атакой). Криптология и состоит из двух частей - криптографии, изучающей способы шифрования и/или проверки подлинности сообщений, и криптоанализа, рассматривающего пути расшифровки и подмены криптограмм. Неустойчивость первых шифров на многие столетия породила атмосферу секретности вокруг работы криптографа, затормозила развитие криптологии как науки.

Так называемая "донаучная" криптография более чем за две тысячи лет полуинтуитивно "нащупала" довольно много интересных решений. Простейшее действие - выполнить подстановку не в алфавитном порядке. Неплохо также переставить символы в сообщении местами (шифры перестановок).

Первым систематическим трудом по криптографии принято считать работу великого архитектора Леона Баттиста Альберти (1404 - 1472 гг.). Период до середины XVII века уже насыщен работами по криптографии и криптоанализу. Интриги вокруг шифрограмм в Европе того времени удивительно интересны. Франсуа Виет (1540 - 1603 гг.), который при дворе короля Франции Генриха IV так успешно занимался криптоанализом (тогда еще не носившим этого гордого названия), что испанский король Филипп II жаловался Папе Римскому на применение французами черной магии. Но все обошлось без кровопролития - при дворе Папы в это время уже служили советники из семейства Ардженти, которых мы сегодня назвали бы криптоаналитиками.

Можно утверждать, что на протяжении веков дешифрованию криптограмм помогает частотный анализ появления отдельных символов и их сочетаний. Вероятности появления отдельных букв в тексте сильно разнятся (для русского языка, например, буква "о" появляется в 45 раз чаще буквы "ф"). Это, с одной стороны, служит основой как для раскрытия ключей, так и для анализа алгоритмов шифрования, а с другой - является причиной значительной избыточности (в информационном смысле) текста на естественном языке. Любая простая подстановка не позволяет спрятать частоту появления символа - как шило из мешка торчат в русском тексте символы, соответствующие буквам "о", "е", "а", "и", "т", "н". Но теория информации и мера избыточности еще не созданы, и для борьбы с врагом криптографа - частотным анализом - предлагается РАНДОМИЗАЦИЯ. Ее автор Карл Фридрих Гаусс (1777 - 1855 гг.) ошибочно полагал, что создал нераскрываемый шифр.

Следующая заметная личность в истории криптологии, которую мы не должны пропустить, - голландец Огюст Керкхофф (1835 - 1903 гг.). Ему принадлежит замечательное "правило Керкхоффа": стойкость шифра должна определяться ТОЛЬКО секретностью ключа. Учитывая время, когда это правило было сформулировано, его можно признать величайшим открытием (до создания систематической теории еще более полувека!). Это правило полагает, что АЛГОРИТМ шифрования НЕ ЯВЛЯЕТСЯ СЕКРЕТНЫМ, а значит, можно вести открытое обсуждение достоинств и недостатков алгоритма. Таким образом, это правило переводит работы по криптологии в разряд ОТКРЫТЫХ научных работ, допускающих дискуссии, публикации и т. п.

**1.3 Криптология в наши дни**

Последнее имя, которое мы назовем в донаучной криптологии, - инженер Жильбер Вернам (G.S. Vernam). В 1926 году он предложил действительно нераскрываемый шифр. Идея шифра состоит в том, чтобы в уравнении (1) для каждого следующего символа выбирать новое значение z. Другими словами, секретный ключ должен использоваться только один раз. Если такой ключ выбирается случайным образом, то, как было строго доказано Шенноном через 23 года, шифр является нераскрываемым. Этот шифр является теоретическим обоснованием для использования так называемых "шифроблокнотов", широкое применение которых началось в годы второй мировой войны. Шифроблокнот содержит множество ключей однократного использования, последовательно выбираемых при шифровании сообщений. Предложение Вернама, однако, не решает задачи секретной связи: вместо способа передачи секретного сообщения теперь необходимо найти способ передачи секретного ключа, РАВНОГО ему ПО ДЛИНЕ, т. е. содержащего столько же символов, сколько имеется в открытом тексте.

В 1949 году статья Клода Шеннона "Теория связи в секретных системах" положила начало научной криптологии. Шеннон показал, что для некоторого "случайного шифра" количество знаков шифротекста, получив которые криптоаналитик при неограниченных ресурсах может восстановить ключ (и раскрыть шифр),

H (Z)/(rlog N), (2)

где H (Z) - энтропия ключа, r - избыточность открытого текста, а N - объем алфавита. По эффективности, с которой архиваторы сжимают текстовые файлы, нам хорошо известно, как велика избыточность обычного текста - ведь их работа и состоит в снижении избыточности (причем только на наиболее легко устраняемой ее части). При избыточности обычного текста порядка 0,75 и использовании 56-битового ключа (такого, как предполагает DES), достаточно 11 символов шифротекста для восстановления ключа при неограниченных ресурсах криптоаналитика.

Строго говоря, соотношение (2) не доказано для произвольного шифра, но верно для известных частных случаев. Из (2) следует замечательный вывод: работу криптоаналитика можно затруднить не только совершенствованием криптосистемы, но и снижением избыточности открытого текста. Более того, если избыточность открытого текста снизить до нуля, то даже короткий ключ даст шифр, который криптоаналитик не сможет раскрыть.

Перед шифрованием информацию следует подвергнуть статистическому кодированию (сжатию, архивации). При этом уменьшится объем информации и ее избыточность, повысится энтропия (среднее количество информации, приходящееся на один символ). Так как в сжатом тексте будут отсутствовать повторяющиеся буквы и слова, дешифрование (криптоанализ) затруднится.

**1.4 Классификация алгоритмов шифрования**

1. Симметричные (с секретным, единым ключом, одноключевые, single-key).

1.1. Потоковые (шифрование потока данных):

* с одноразовым или бесконечным ключом (infinite-key cipher);
* с конечным ключом (система Вернама - Vernam);
* на основе генератора псевдослучайных чисел (ПСЧ).

1.2. Блочные (шифрование данных поблочно):

1.2.1. Шифры перестановки (permutation, P-блоки);

1.2.2. Шифры замены (подстановки, substitution, S-блоки):

* моноалфавитные (код Цезаря);
* полиалфавитные (шифр Видженера, цилиндр Джефферсона, диск Уэтстоуна, Enigma);

2. Асимметричные (с открытым ключом, public-key):

* Диффи-Хеллман DH (Diffie, Hellman);
* Райвест-Шамир-Адлeман RSA (Rivest, Shamir, Adleman);
* Эль-Гамаль (ElGamal).

Кроме того, есть разделение алгоритмов шифрования на собственно шифры (ciphers) и коды (codes). Шифры работают с отдельными битами, буквами, символами. Коды оперируют лингвистическими элементами (слоги, слова, фразы).

Глава II. Рассмотрение алгоритмов

**2.1 Симметричные алгоритмы шифрования**

Симметричные алгоритмы шифрования (или криптография с секретными ключами) основаны на том, что отправитель и получатель информации используют один и тот же ключ. Этот ключ должен храниться в тайне и передаваться способом, исключающим его перехват.

Обмен информацией осуществляется в 3 этапа:

* отправитель передает получателю ключ (в случае сети с несколькими абонентами у каждой пары абонентов должен быть свой ключ, отличный от ключей других пар);
* отправитель, используя ключ, зашифровывает сообщение, которое пересылается получателю;
* получатель получает сообщение и расшифровывает его.

Если для каждого дня и для каждого сеанса связи будет использоваться уникальный ключ, это повысит защищенность системы.

**2.1.1 Потоковые шифры**

В потоковых шифрах, т. е. при шифровании потока данных, каждый бит исходной информации шифруется независимо от других с помощью гаммирования.

Гаммирование - наложение на открытые данные гаммы шифра (случайной или псевдослучайной последовательности единиц и нулей) по определенному правилу. Обычно используется "исключающее ИЛИ", называемое также сложением по модулю 2 и реализуемое в ассемблерных программах командой XOR. Для расшифровывания та же гамма накладывается на зашифрованные данные.

При однократном использовании случайной гаммы одинакового размера с зашифровываемыми данными взлом кода невозможен (так называемые криптосистемы с одноразовым или бесконечным ключом). В данном случае "бесконечный" означает, что гамма не повторяется.

В некоторых потоковых шифрах ключ короче сообщения. Так, в системе Вернама для телеграфа используется бумажное кольцо, содержащее гамму. Конечно, стойкость такого шифра не идеальна.

Понятно, что обмен ключами размером с шифруемую информацию не всегда уместен. Поэтому чаще используют гамму, получаемую с помощью генератора псевдослучайных чисел (ПСЧ). В этом случае ключ - порождающее число (начальное значение, вектор инициализации, initializing value, IV) для запуска генератора ПСЧ. Каждый генератор ПСЧ имеет период, после которого генерируемая последовательность повторяется. Очевидно, что период псевдослучайной гаммы должен превышать длину шифруемой информации.

Генератор ПСЧ считается корректным, если наблюдение фрагментов его выхода не позволяет восстановить пропущенные части или всю последовательность при известном алгоритме, но неизвестном начальном значении.

При использовании генератора ПСЧ возможны несколько вариантов

1. Побитовое шифрование потока данных. Цифровой ключ используется в качестве начального значения генератора ПСЧ, а выходной поток битов суммируется по модулю 2 с исходной информацией. В таких системах отсутствует свойство распространения ошибок.

2. Побитовое шифрование потока данных с обратной связью (ОС) по шифртексту. Такая система аналогична предыдущей, за исключением того, что шифртекст возвращается в качестве параметра в генератор ПСЧ. Характерно свойство распространения ошибок. Область распространения ошибки зависит от структуры генератора ПСЧ.

3. Побитовое шифрование потока данных с ОС по исходному тексту. Базой генератора ПСЧ является исходная информация. Характерно свойство неограниченного распространения ошибки.

4. Побитовое шифрование потока данных с ОС по шифртексту и по исходному тексту.

**2.1.2 Блочные шифры**

При блочном шифровании информация разбивается на блоки фиксированной длины и шифруется поблочно. Блочные шифры бывают двух основных видов:

* шифры перестановки (transposition, permutation, P-блоки);
* шифры замены (подстановки, substitution, S-блоки).

Шифры перестановок переставляют элементы открытых данных (биты, буквы, символы) в некотором новом порядке. Различают шифры горизонтальной, вертикальной, двойной перестановки, решетки, лабиринты, лозунговые и др.

Шифры замены заменяют элементы открытых данных на другие элементы по определенному правилу. Paзличают шифры простой, сложной, парной замены, буквенно-слоговое шифрование и шифры колонной замены. Шифры замены делятся на две группы:

* моноалфавитные (код Цезаря);
* полиалфавитные (шифр Видженера, цилиндр Джефферсона, диск Уэтстоуна, Enigma).

В моноалфавитных шифрах замены буква исходного текста заменяется на другую, заранее определенную букву. Например в коде Цезаря буква заменяется на букву, отстоящую от нее в латинском алфавите на некоторое число позиций. Очевидно, что такой шифр взламывается совсем просто. Нужно подсчитать, как часто встречаются буквы в зашифрованном тексте, и сопоставить результат с известной для каждого языка частотой встречаемости букв.

В полиалфавитных подстановках для замены некоторого символа исходного сообщения в каждом случае его появления последовательно используются различные символы из некоторого набора. Понятно, что этот набор не бесконечен, через какое-то количество символов его нужно использовать снова. В этом слабость чисто полиалфавитных шифров.

В современных криптографических системах, как правило, используют оба способа шифрования (замены и перестановки). Такой шифратор называют составным (product cipher). Oн более стойкий, чем шифратор, использующий только замены или перестановки.

**2.2 Асимметричные алгоритмы шифрования**

В асимметричных алгоритмах шифрования (или криптографии с открытым ключом) для зашифровывания информации используют один ключ (открытый), а для расшифровывания - другой (секретный). Эти ключи различны и не могут быть получены один из другого.

Схема обмена информацией такова:

* получатель вычисляет открытый и секретный ключи, секретный ключ хранит в тайне, открытый же делает доступным (сообщает отправителю, группе пользователей сети, публикует);
* отправитель, используя открытый ключ получателя, зашифровывает сообщение, которое пересылается получателю;
* получатель получает сообщение и расшифровывает его, используя свой секретный ключ.

**2.2.1 Алгоритм Диффи-Хелмана**

Алгоритм Диффи-Хелмана (Whitfield Diffie и Martin Hellman, 1976 год) использует функцию дискретного возведения в степень и похож на метод Эль-Гамаля.

Сначала генерируются два больших простых числа **n** и **q**. Эти два числа не обязательно хранить в секрете. Далее один из партнеров **P1** генерирует случайное число **x** и посылает другому участнику будущих обменов **P2** значение **A = qx mod n**

По получении **А** партнер **P2** генерирует случайное число **у** и посылает **P2** вычисленное значение **B = qy mod n**

Партнер **P1**, получив **В**, вычисляет **Kx = Bx mod n**, а партнер **P2** вычисляет **Ky = Ay mod n**. Алгоритм гарантирует, что числа **Ky** и **Kx**равны и могут быть использованы в качестве секретного ключа для шифрования. Ведь даже перехватив числа А и В, трудно вычислить **Kx** или **Ky**.

Алгоритм Диффи-Хелмана, обеспечивая конфиденциальность передачи ключа, не может гарантировать того, что он прислан именно тем партнером, который предполагается. Для решения этой проблемы был предложен протокол **STS** (station-to-station). Этот протокол для идентификации отправителя использует технику электронной подписи. Подпись шифруется общим секретным ключом, после того как он сформирован. Подпись включает в себя идентификаторы как P1, так и P2.

**2.2.2 RSA**

Защищен патентом США N 4405829. Разработан в 1977 году в Массачусетском технологическом институте (США). Получил название по первым буквам фамилий авторов (Rivest, Shamir, Adleman). Криптостойкость основана на вычислительной сложности задачи разложения большого числа на простые множители.

Алгоритм RSA предполагает, что посланное закодированное сообщение может быть прочитано адресатом и только им. В этом алгоритме используется два ключа - **открытый и секретный**. Данный алгоритм привлекателен также в случае, когда большое число субъектов (N) должно общаться по схеме все-со-всеми. В случае симметричной схемы шифрования каждый из субъектов каким-то образом должен доставить свои ключи всем остальным участникам обмена, при этом суммарное число используемых ключей будет достаточно велико при большом значении N. Применение асимметричного алгоритма требует лишь рассылки открытых ключей всеми участниками, суммарное число ключей равно N.

Сообщение представляется в виде числа **M**. Шифрование осуществляется с помощью общедоступной функции **f(M),** и только адресату известно, как выполнить операцию **f-1**. Адресат выбирает два больших простых (prime) числа **p** и **q**, которые делает секретными. Он объявляет **n=pq** и число **d**, c **(d,p-1)=(d,q-1)=1** (один из возможных способов выполнить это условие, выбрать **d** больше чем **p/2** и **q/2**). Шифрование производится по формуле:

**f(M) ≡ Md mod n,**

где **M** и **f(M)** оба ≤ **n-1**. Как было показано, может быть вычислено за разумное время, даже если **M**, **d** и **n** содержит весьма большое число знаков. Адресат вычисляет **M** на основе **Md**, используя свое знание **p** и **q.** В соответствие со следствием , если

**dc ≡ (p-1)1, тогда (Md)e ≡ p1.**

Исходный текст **M** получается адресатом из зашифрованного F(M) путем преобразования: M = (F(M))e (mod pq). Здесь как исходный текст, так и зашифрованный рассматриваются как длинные двоичные числа.

Аналогично **(Md)e ≡ qM, если dc ≡ (q-1)1**. **e** удовлетворяет этим двум условиям, если **cd ≡ (p-1) (q-1)1**. Теорема 1 гласит, что мы можем позволить e=x, когда x является решением уравнения **dx + (p-1)(q-1)y = 1.**

Так как (**Md)e** - M делимо на **p** и **q**, оно делимо и на **pq**, следовательно, мы можем определить **M,** зная **Md,** вычислив его значение в степени e и определив остаток от деления на **pq**. Для соблюдения секретности важно, чтобы, зная **n,** было нельзя вычислить **p и q.** Если **n** содержит 100 цифр, подбор шифра связан с перебором ~1050 комбинаций. Данная проблема изучается уже около 100 лет. RSA-алгоритм запатентован (20 сентября 1983, действует до 2000 года).

Теоретически можно предположить, что возможно выполнение операции **f-1,** не вычисляя **p** и **q**. Но в любом случае задача эта не проста и разработчики считают ее трудно факторизуемой.

Предположим, что мы имеем зашифрованный текст **f(M)** и исходный текст **M,** и мы хотим найти значения **p** и **q**. Нетрудно показать, что таких исходных данных для решения задачи недостаточно - надо знать все возможные значения **Mi**.

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

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

**2.2.3 ElGamal**

Разработан в 1985 году. Назван по фамилии автора - Эль-Гамаль. Алгоритм Эль-Гамаля может использоваться для формирования электронной подписи или для шифрования данных. Он базируется на трудности вычисления дискретного логарифма. Для генерации пары ключей сначала берется простое число **p** и два случайных числа **g** и **x**, каждое из которых меньше **p**. Затем вычисляется:

**y = gx mod p**

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

**yaab mod p = gM mod p.**

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

**M = b/ax mod p**

**2.3 Сравнение cимметричных и аcимметричных алгоритмов шифрования**

В асимметричных системах необходимо применять длинные ключи (512 битов и больше). Длинный ключ резко увеличивает время шифрования. Кроме того, генерация ключей весьма длительна. Зато распределять ключи можно по незащищенным каналам.

В симметричных алгоритмах используют более короткие ключи, т. е. шифрование происходит быстрее. Но в таких системах сложно распределение ключей.

Поэтому при проектировании защищенной системы часто применяют и cимметричные, и аcимметричные алгоритмы. Так как система с открытыми ключами позволяет распределять ключи и в симметричных системах, можно объединить в системе передачи защищенной информации асимметричный и симметричный алгоритмы шифрования. С помощью первого рассылать ключи, вторым же - собственно шифровать передаваемую информацию

Обмен информацией можно осуществлять следующим образом:

* получатель вычисляет открытый и секретный ключи, секретный ключ хранит в тайне, открытый же делает доступным;
* отправитель, используя открытый ключ получателя, зашифровывает сеансовый ключ, который пересылается получателю по незащищенному каналу;
* получатель получает сеансовый ключ и расшифровывает его, используя свой секретный ключ;
* отправитель зашифровывает сообщение сеансовым ключом и пересылает получателю;
* получатель получает сообщение и расшифровывает его.

Надо заметить, что в правительственных и военных системах связи используют лишь симметричные алгоритмы, так как нет строго математического обоснования стойкости систем с открытыми ключами, как, впрочем, не доказано и обратное.

**2.4 Использование инструментов криптографии в Delphi-приложениях**

**2.4.1 CryptoAPI**

Криптографические функции являются частью операционной системы Windows, и обратится к ним можно посредством интерфейса CryptoAPI. Основные возможности доступны еще с Windows 95, но со временем они расширялись. Описание функций CryptoAPI можно найти в MSDN, или в справочном файле к Delphi. Функции содержаться в библиотеках advapi32.dll и crypt32.dll.

**2.4.2 Подключение к криптопровайдеру. Контейнеры ключей**

Первая функция, которую мы рассмотрим, будет

function CryptAcquireContext(phProv :PHCRYPTPROV;

pszContainer :LPAWSTR;

pszProvider :LPAWSTR;

dwProvType :DWORD;

dwFlags :DWORD) :BOOL; stdcall;

В большинстве случаев, работа с криптографическими возможностями Windows начинается с вызова именно этой функции, которая выполняет подключение к криптопровайдеру и возвращает его дескриптор в параметре phProv. Криптопровайдер представляет собой dll, независимый программный модуль, который фактически исполняет криптографические алгоритмы. Криптопровайдеры бывают различные и отличаются составом функций (например, некоторые криптопровайдеры ограничиваются лишь цифровыми подписями), используемыми алгоритмами (некоторые шифруют алгоритмом RC2, другие - DES) и другими возможностями. В каждой операционной системе свой состав криптопровайдеров, однако в каждой присутствует Microsoft Base Cryptographic Provider v1.0. При вызове функции CryptAcquireContext, необходимо указать имя провайдера и его тип (соответственно в параметрах pszProvider и dwProvType). Тип провайдера определяет состав функций и поддерживаемые криптоалгоритмы, например:

**Тип PROV\_RSA\_FULL**

* Обмен ключами - алгоритм RSA
* Цифровая подпись - алгоритм RSA
* Шифрование - алгоритм RC2 и RC4
* Хэширование - алгоритмы MD5 и SHA

**Тип PROV\_RSA\_SIG**

* Обмен ключами - не поддерживается
* Цифровая подпись - алгоритм RSA
* Шифрование - не поддерживается
* Хэширование - алгоритмы MD5 и SHA

Microsoft Base Cryptographic Provider v1.0 относится к типу PROV\_RSA\_FULL и для этого типа используется по умолчанию (если в параметре pszProvider указать nil). В параметре pszContainer необходимо указать имя контейнера ключей, который мы собираемся использовать. Дело в том, что каждый криптопровайдер содержит базу данных, в которой хранятся ключи пользователей. Эти ключи группируются в контейнерах. Сохраняются только ключевые пары для асимметричных алгоритмов, сеансовые ключи не сохраняются, так как их не рекомендуют использовать повторно. Таким образом, каждый контейнер имеет имя и содержит по одному ключу (точнее паре открытый-закрытый ключ) для цифровой подписи и обмена ключами. В зависимости от криптопровайдера, база данных может храниться в файлах, реестре или в каких-либо аппаратных средствах, но это не влияет на работу программиста с контейнерами ключей. Если в качестве параметра pszContainer указать nil, то будет использоваться контейнер ключей, название которого совпадает именем пользователя, под которым был осуществлен вход в систему. Но так делать не рекомендуется: дело в том, что если два приложения использует один и тот же контейнер, одно из них может изменить или уничтожить ключи, необходимые для корректной работы другого приложения. Поэтому рекомендуют использовать контейнеры, имена которых совпадает с именем приложения.

Параметр dwFlags может быть нулевым или принимать одно из следующих значений:

CRYPT\_VERIFYCONTEXT - этот флаг предназначен для приложений, которые не должны иметь доступ к закрытым ключам контейнера. Такие приложения могут обращаться только к функциям хеширования, проверки цифровой подписи или симметричного шифрования. В этом случае параметр pszContainer должен быть равен nil.

CRYPT\_NEWKEYSET - создает новый контейнер ключей, но сами ключи не создаются.

CRYPT\_DELETEKEYSET - удаляет контейнер вместе с хранящимися там ключами. Если задан этот флаг, то подключение к криптопровайдеру не происходит и параметр phProv неопределен.

CRYPT\_MACHINE\_KEYSET - по умолчанию контейнеры ключей сохраняются как пользовательские. Для основных криптопровайдеров это означает, что контейнеры ключей сохраняются в пользовательских профилях. Этот флаг можно устанавливать в комбинации с другими, чтобы указать, что контейнер является машинным, то есть хранится в профиле All Users.

В случае успеха, функция возвращает true, в противном случае - false. GetLastError вернет код ошибки.

function CryptReleaseContext(hProv :HCRYPTPROV;

dwFlags :DWORD) :BOOL; stdcall;

Освобождает контекст криптопровайдера и контейнера ключей. hProv - дескриптор криптопровайдера, полученный при вызове CryptAcquireContext. dwFlags - зарезервирован и должен равняться нулю.

В случае успеха, функция возвращает true, в противном случае - false. GetLastError вернет код ошибки.

Приведем пример работы с этими функциями:

uses Wcrypt2;

...

procedure CryptProc;

var

Prov: HCRYPTPROV;

begin

CryptAcquireContext(@Prov,nil,nil,PROV\_RSA\_FULL,CRYPT\_VERIFYCONTEXT);

// Работаем с функциями CryptoAPI

...

CryptReleaseContext(Prov,0);

end;

**2.4.3 Шифрование на основе пользовательских данных или пароля**

Для шифрования в CryptoAPI используются симметричные алгоритмы, ключ для которых может быть получен двумя путями: случайным образом или на основе каких-либо пользовательских данных, например пароля. Причем к последнему варианту генерации ключа есть одно важное требование: при использовании одних и тех же паролей должны получаться идентичные ключи. Такая возможность предусмотрена в CryptoAPI.

function CryptDeriveKey(hProv :HCRYPTPROV;

Algid :ALG\_ID;

hBaseData :HCRYPTHASH;

dwFlags :DWORD;

phKey :PHCRYPTKEY) :BOOL; stdcall;

В параметре hProv нужно указать дескриптор провайдера, полученный с помощью CryptAcquireContext. Algid - идентификатор алгоритма, для которого генерируется ключ. Для Microsoft Base Cryptographic Provider может принимать следующие значения: CALG\_RC2 и CALG\_RC4. Пользовательские данные (пароль) предварительно хэшируются и дескриптор хэш-объекта передается в функцию в качестве параметра hBaseData. Старшие 16 бит параметра dwFlags могут содержать размер ключа в битах или быть нулевыми (в этом случае будет создан ключ с размером по умолчанию). Младшие 16 бит могут быть нулевыми или принимать следующие значения или их комбинации: CRYPT\_EXPORTABLE, CRYPT\_CREATE\_SALT, CRYPT\_USER\_PROTECTED, CRYPT\_UPDATE\_KEY. К первым двум мы еще вернемся, а со смыслом остальных вы можете ознакомиться самостоятельно. В параметре phKey возвращается дескриптор созданного ключа.

В случае успеха, функция возвращает true, в противном случае - false. GetLastError вернет код ошибки.

Когда ключ есть, можно приступать непосредственно к шифрованию. Для этого нам понадобятся функции CryptEncrypt и CryptDecrypt.

function CryptEncrypt(hKey :HCRYPTKEY;

hHash :HCRYPTHASH;

Final :BOOL;

dwFlags :DWORD;

pbData :PBYTE;

pdwDataLen :PDWORD;

dwBufLen :DWORD) :BOOL; stdcall;

В параметре hKey передается дескриптор ключа, необходимый для шифрования. Этот ключ также определяет алгоритм шифрования. Параметр hHash используется, если данные одновременно шифруются и хэшируются (шифроваться и хэшироваться будут исходные данные). В этом случае в параметре hHash передается дескриптор заранее созданного хэш-объекта. Эту возможность удобно использовать, если необходимо одновременно зашифровать и подписать сообщение. Иначе этот параметр следует установить в ноль. Параметр Final следует установить в true, если переданный в функцию блок данных является единственным или последним. В этом случае он будет дополнен до необходимого размера. Параметр dwFlags не используется в Microsoft Base Cryptographic Provider и на его месте следует указать ноль. pbData - указатель на буфер, в котором содержаться данные для зашифрования. Зашифрованыые данные помещаются в тот же буфер. pdwDataLen - размер данных, которые будут зашифрованы. dwBufLen - размер выходного буфера, для блочных шифров может быть больше, чем pdwDataLen. Узнать необходимый размер, можно передав в параметре pbData nil, в параметре pdwDataLen - размер данных, которые необходимо зашифровать, а в параметре dwBufLen - что угодно, например ноль. После такого вызова, необходимый размер буфера будет содержаться в параметре pdwDataLen (именно pdwDataLen, а не dwBufLen, немного нелогично, ну да ладно). Чтобы не было путаницы, приведем пример:

var

Message: String;

BufLen, DataLen: DWORD;

...

begin

...

Message:='Hello World!';

BufLen:=Length(Message);

DataLen:=Length(Message);

// Вычисляем необходимый размер выходного буфера

CryptEncrypt(Key,0,true,0,nil,@BufLen,0);

// Выделяем память для буфера и шифруем

SetLength(Message,BufLen);

CryptEncrypt(Key,0,true,0,PByte(Message),@DataLen,BufLen);

Теперь, рассмотрим функцию, которая позволяет расшифровать сообщение.

function CryptDecrypt(hKey :HCRYPTKEY;

hHash :HCRYPTHASH;

Final :BOOL;

dwFlags :DWORD;

pbData :PBYTE;

pdwDataLen :PDWORD) :BOOL; stdcall;

В параметр pdwDataLen нужно передать число байт шифротекста, а после вызова в него будет помещена длина открытого сообщения. Если используется параметр hHash, то данные после расшифровки хэшируются. Это удобно использовать, если нужно одновременно расшифровать сообщение и проверить подпись.

После того, как работа с ключом закончена, необходимо освободить дескриптор:

function CryptDestroyKey(hKey :HCRYPTKEY) :BOOL; stdcall;

Если hKey относится к сеансовому ключу или импортированному открытому ключу (об этом ниже), то дескриптор освобождается, а ключ уничтожается. Если hKey относится к паре открытый/закрытый ключ, то дескриптор освобождается, а ключевая пара сохраняется в контейнере ключей.

Только что мы рассмотрели случай, когда для зашифровки и расшифровки сообщения отправитель и получатель использовали пароль, известный только им. Сейчас рассмотрим другой: отправитель генерирует ключ случайно и передает его получателю в зашифрованном виде вместе с сообщением. При этом для шифрования сеансового ключа используется открытый ключ получателя. А где отправитель его возьмет?

Как уже было сказано, при создании ключевого контейнера с помощью функции CryptAcquireContext, ключи в контейнере не создаются, их нужно сгенерировать отдельно. Рассмотрим функцию:

function CryptGenKey(hProv :HCRYPTPROV;

Algid :ALG\_ID;

dwFlags :DWORD;

phKey :PHCRYPTKEY) :BOOL; stdcall;

Функция предназначена для генерации случайных сеансовых ключей и ключевых пар. Параметры этой функции аналогичны одноименным параметрам функции CryptDeriveKey, за исключением того, что Algid может также принимать значения AT\_KEYEXCHANGE и AT\_SIGNATURE. В этом случае будут сгенерированы ключевые пары соответственно для обмена ключами и цифровой подписи. Создание нового ключевого контейнера должно выглядеть примерно так:

uses Wcrypt2;

...

var

Prov: HCRYPTPROV;

ExchangeKey, SignKey: HCRYPTKEY;

begin

CryptAcquireContext(@Prov,'My\_Container',nil,PROV\_RSA\_FULL,CRYPT\_NEWKEYSET);

// Создаем ключевые пары

CryptGenKey(Prov,AT\_KEYEXCHANGE,0,@ExchangeKey);

CryptGenKey(Prov,AT\_SIGNATURE,0,@SignKey);

// Работаем с функциями CryptoAPI

...

// Освобождаем дескрипторы ключевых пар. Сами ключи сохраняются в контейнере

CryptDestroyKey(SignKey);

CryptDestroyKey(ExchangeKey);

CryptReleaseContext(Prov,0);

end;

Созданные таким образом ключевые пары, впоследствии можно извлечь из контейнера, воспользовавшись функцией

function CryptGetUserKey(hProv :HCRYPTPROV;

dwKeySpec :DWORD;

phUserKey :PHCRYPTKEY) :BOOL; stdcall;

Параметр dwKeySpec может принимать два значения: AT\_KEYEXCHANGE и AT\_SIGNATURE, значения которых очевидны. Дескриптор ключа возвращается в параметре phUserKey.

Теперь ответим на вопрос, как отправитель сможет передать получателю свою открытую часть ключа.

function CryptExportKey(hKey :HCRYPTKEY;

hExpKey :HCRYPTKEY;

dwBlobType :DWORD;

dwFlags :DWORD;

pbData :PBYTE;

pdwDataLen :PDWORD) :BOOL; stdcall;

Функция позволяет экспортировать ключ в двоичный буфер, который впоследствии можно будет сохранить в файл и передать кому-либо. В параметре hKey должен содержаться дескриптор экспортируемого ключа. Экспортировать можно не только открытые ключи, а также ключевые пары целиком и сеансовые ключи. В последних двух случаях, ключи и ключевые пары должны быть созданы функциями CryptGenKey или CryptDeriveKey с параметрами dwFlags равными CRYPT\_EXPORTABLE. Открытые же ключи всегда экспортируемы. Сеансовые ключи и ключевые пары экспортируются только в зашифрованном виде. Параметр hExpKey определяет ключ, которым они будут зашифрованы. Если экспортируется открытая часть ключа, то этот параметр следует установить в ноль, если экспортируется ключевая пара целиком, то здесь обычно передают дескриптор сеансового ключа (обычно полученный с помощью CryptDeriveKey), которым пара будет зашифрована, если экспортируется сеансовый ключ, то обычно он шифруется открытым ключом получателя (обычно используется ключ обмена, но никто не запрещает использовать ключ подписи). Параметр dwBlobType определяет тип экспортируемого ключа и может принимать следующие значения: SIMPLEBLOB - сеансовый ключ, PUBLICKEYBLOB - открытый ключ, PRIVATEKEYBLOB - ключевая пара целиком. Существуют и другие значения, но они не поддерживаются стандартным криптопровайдером. Параметр dwFlags для Microsoft Base Cryptographic Provider должен быть равен нулю. pbData - буфер, куда будут скопированы данные, pdwDataLen - размер этого буфера. Если он заранее не известен, то можно указать в качестве параметра pbData nil, и в pdwDataLen будет получен необходимый размер.

Вот пример экспорта открытого ключа:

procedure ExportPublicKey(FileName: TFileName);

var

Prov: HCRYPTPROV;

SignKey: HCRYPTKEY;

Stream: TMemoryStream;

BufSize: DWORD;

begin

CryptAcquireContext(@Prov,'My\_Container',nil,PROV\_RSA\_FULL,0);

CryptGetUserKey(Prov,AT\_SIGNATURE,@SignKey);

Stream:=TMemoryStream.Create;

CryptExportKey(SignKey,0,PUBLICKEYBLOB,0,nil,@BufSize);

Stream.SetSize(BufSize);

CryptExportKey(SignKey,0,PUBLICKEYBLOB,0,PByte(Stream.Memory),@BufSize);

Stream.SaveToFile(FileName);

Stream.Free;

CryptDestroyKey(SignKey);

CryptReleaseContext(Prov,0);

end;

Импорт ключа осуществляется с помощью функции

function CryptImportKey(hProv :HCRYPTPROV;

pbData :PBYTE;

dwDataLen :DWORD;

hPubKey :HCRYPTKEY;

dwFlags :DWORD;

phKey :PHCRYPTKEY) :BOOL; stdcall;

В параметре hPubKey необходимо передать дескриптор ключа, которым будет расшифрован импортированный ключ. Если импортируется ключевая пара целиком, то параметр dwFlags можно установить в CRYPT\_EXPORTABLE, тогда импортированная пара может быть впоследствии также экспортирована. В параметре phKey вернется дескриптор полученного ключа. Если это ключевая пара, то она будет сохранена в контейнере.

Вот пример импорта открытого ключа:

function ImportPublicKey(FileName: TFileName): HCRYPTKEY;

var

Prov: HCRYPTPROV;

Stream: TMemoryStream;

begin

Stream:=TMemoryStream.Create;

Stream.LoadFromFile(FileName);

CryptImportKey(Prov,PByte(Stream.Memory),Stream.Size,0,0,@Result);

Stream.Free;

end;

Теперь, воспользовавшись этой информацией, мы без труда сможем восстановить пропущенные фрагменты в функции проверки цифровой подписи, описанной ранее.

Итак, как же передать собеседнику зашифрованное сообщение:

* Получатель экспортирует свой открытый ключ обмена в файл и передает его отправителю сообщения.
* Отправитель генерирует случайный сеансовый ключ и шифрует им сообщение.
* Отправитель импортирует открытый ключ обмена получателя, экспортирует сеансовый ключ, шифруя его полученным ключом обмена (ключ обмена в параметре hExpKey).
* Зашифрованное сообщение передается вместе с зашифрованным сеансовым ключом - так называемый цифровой конверт.
* Получатель импортирует сеансовый ключ, расшифровывая его своим закрытым ключом обмена (его можно получить, вызвав CryptGetUserKey) и с помощью сеансового ключа расшифровывает сообщение.

Говоря о сеансовых ключах, используемых в Microsoft Base Cryptographic Provider нужно упомянуть об одной неприятности: до начала 2000 года действовал запрет на экспорт программного обеспечения, использующего средства "сильной криптографии" за пределами США и Канады. По этой причине в базовом криптопровайдере не поддерживаются ключи для симметричных алгоритмов длиной более 40 бит. Ключи длиной 56 бит разрешалось использовать только заграничным отделениям американских компаний. Для алгоритмов RC2 и RC4 рекомендуемая длина ключа должна составлять 128 бит, поэтому недостающее количество бит заполняется нулями либо случайными значениями, которые должны передаваться открыто. Надежность защиты из-за этого, разумеется, сильно страдает. В состав Windows XP входит Microsoft Enhanced Cryptographic Provider, в котором этой проблемы нет, но при использовании базового криптопровайдера, необходимо дополнять ключ до нужной длины, используя т.н. солт-значения (salt-values). Сгенерировать salt-value и внести его в ключ можно несколькими способами, но самый простой и очевидный - при вызове CryptGenKey или CryptDeriveKey передать в параметре dwFlags значение CRYPT\_CREATE\_SALT, примерно так:

CryptGenKey(Prov,CALG\_RC2,CRYPT\_EXPORTABLE or CRYPT\_CREATE\_SALT,@Key);

При экспорте ключа солт-значение не сохраняется, о нем должен позаботиться сам программист.

var

SaltLen: DWORD;

Stream: TMemoryStream;

...

begin

...

// Определяем размер буфера для солт-значения

CryptGetKeyParam(Key,KP\_SALT,nil,@SaltLen,0);

// Сохраняем его в файл

Stream:=TMemoryStream.Create;

Stream.SetSize(SaltLen);

CryptGetKeyParam(Key,KP\_SALT,PByte(Stream.Memory),@SaltLen,0);

Stream.SaveToFile('Salt.dat');

Stream.Free;

...

Сохраненное таким образом солт-значение необходимо передать вместе с сеансовым ключом, а на приемной стороне "вживить" его туда снова.

var

Stream: TMemoryStream;

...

begin

...

Stream:=TMemoryStream.Create;

Stream.LoadFromFile('Salt.dat');

CryptSetKeyParam(Key,KP\_SALT,PByte(Stream.Memory),Stream.Size);

Stream.Free;

...

**2.5 Постановка задачи**

Разработать программу шифрующую и дешифрующую введенный пользователем текст. Обеспечить ввод и вывод информации с помощью файлов. Обеспечить достаточную криптоустойчивость шифра.

**2.6 Реализация задачи**

## 

## 2.6.1 Краткая характеристика среды Delphi 7

Среда программирования **Delphi 7** позволяетреализовать поставленную задачу со всеми необходимыми требованиями. В среде используется язык **Object Pascal. Delphi 7** является объектно-ориентированной средой, что упрощает создание единообразного интерфейса и ввод-вывод информации из файла.

## 

## 2.6.2 Алгоритм решения задачи

Программа представляет собой шифровщик и дешифровщик одновременно.

Используется метод симметрического кодирования, то есть кодирование и декодирование осуществляется с помощью одного ключа. Для обеспечения криптоустойчивости необходим длинный ключ. Пользователь вводит 30-символьный код. Блоками по 10 цифр. Из них генерируется код следующим образом:

Берутся первые цифры из 1-го, 2-го,3-го блоков, и в зависимости от 3-го блока над ними производятся операции сложения или умножения. Затем берется следующая цифра 3-го блока, после полного завершения цикла по 3-му блоку берется следующая цифра из второго блока и т.д. Гаммирование завершается после прохождения по циклу последней цифры 1-го блока. Результатом является код состоящий из 1000 независимых, неповторяющихся и незакономерных цифр. Шифр возможно вскрыть только полным перебором, что обеспечивает достаточную криптоустойчивость.

### **Модули программы**

Программа имеет несколько стандартных основных модулей: вызов справки, сохранение в файл, открытие файла, генерация произвольного ключа, проверка пароля, которые не будут упоминаться в дальнейшем.

Опишем лишь модули отвечающие непосредственно за шифрование.

### **Модуль шифровки/дешифровки**

Взятие кодовых цифр из блоков

Чтение отдельного символа исх. текста

Кодирование

символа

Конец исх. текста

Сохр. результата

s1:=memo1.Text;

k:=length(s);

k1:=length(s1);

k3:=length(s3);

k5:=length(s5);

m:=1;

m3:=1; m5:=1;

for i:=1 to k1 do

begin

q:=false;

i5:=strtoint(s[m]);

i2:=strtoint(s3[m3]);

i3:=strtoint(s5[m5]);

i1:=kod(i5,i2,i3);

if i1=0 then i1:=10;

case s1[i] of

'a'..'z':begin c1:=perevod(s1[i],i1,96,122); q:=true; end;

'A'..'Z':begin c1:=perevod(s1[i],i1,64,90); q:=true; end;

'А'..'Я':begin c1:=perevod(s1[i],i1,191,223); q:=true; end;

'а'..'я':begin c1:=perevod(s1[i],i1,223,255); q:=true; end;

end;

if q=true then

begin

{if ord(s1[i])+i1>255 then }

s2:=s2+c1;

{ else

s2:=s2+chr(ord(s1[i])+i1);}

end

else s2:=s2+s1[i];

q1:=false;

if m5=k5 then begin m5:=1; q1:=true; end

else inc(m5);

q2:=false;

if (q1=true) then

begin

inc(m3);

end;

if m3>=k3 then begin m3:=1; q2:=true end;

if (q2=true) then m:=m+1;

if m>=k then m:=1;

end;

{delete(s2,k1-1,2)}

memo2.Text:=s2;

### **Процедура кодирования символа**

begin

if abs(i1)>y-x+1 then

repeat

i1:=i1 mod (y-x+1);

until i1<(y-x+1);

if ord(c)+i1>y then

perevod:=chr(x+(ord(c)+i1-y))

else

perevod:=chr(ord(c)+i1)

end;

## 2.6.3 Таблица сообщений

В ходе работы пользователь может получить сообщение:

«Пароль не подтвержден» - следует убедиться в правильности пароля в дублирующих и основных полях.

**Заключение**

Обоснованный выбор той или иной системы защиты должен опираться на какие-то критерии эффективности. К сожалению, до сих пор не разработаны подходящие методики оценки эффективности криптографических систем.

Наиболее простой критерий такой эффективности - вероятность раскрытия ключа или мощность множества ключей (М). По сути это то же самое, что и кpиптостойкость. Для ее численной оценки можно использовать также и сложность раскрытия шифра путем перебора всех ключей. Однако, этот критерий не учитывает других важных требований к криптосистемам:

* невозможность раскрытия или осмысленной модификации информации на основе анализа ее структуры,
* совершенство используемых протоколов защиты,
* минимальный объем используемой ключевой информации,
* минимальная сложность реализации (в количестве машинных операций), ее стоимость,
* высокая оперативность.

В любом случае выбранный комплекс кpиптогpафических методов должен сочетать как удобство, гибкость и оперативность использования, так и надежную защиту от злоумышленников циркулирующей в ИС информации. Поэтому на настоящий момент наиболее оптимальны смешанные криптосиситемы, в которых текст кодируется симметрически, а ключ кодируется ассиметрически и помещается вместе с кодированным текстом.

Шифрование информации не является панацеей. Его следует рассматривать только как один из методов защиты информации и применять обязательно в сочетании с законодательными, организационными и другими мерами.

**Литература**

1. Водолазский В. Коммерческие системы шифрования: основные алгоритмы и их реализация. Часть 1. // Монитор. - 1992. - N 6-7. - c. 14 - 19.

2. Игнатенко Ю.И. Как сделать так, чтобы?.. // Мир ПК. - 1994. - N 83. Ковалевский В., Максимов В. Криптографические методы. // КомпьютерПресс. - 1993. - N 5. - c. 31 - 34.

4. Мафтик С. Механизмы защиты в сетях ЭВМ. - М.: Мир, 1993.

5. Спесивцев А.В., Вегнер В.А., Крутяков А.Ю. и др. Защита информации в персональных ЭВМ. - M.: Радио и связь, 1992.

6. Сяо Д., Керр Д., Мэдник С. Защита ЭВМ. - М.: Мир, 1982.

7. Шмелева А. Грим - что это ? // Hard'н'Soft. - 1994. - N 5.

8. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования.