ЧАСТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«СТАВРОПОЛЬСКИЙ МНОГОПРОФИЛЬНЫЙ КОЛЛЕДЖ»

**«**Применение криптографических средств защиты информации**»**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим занятиям

для обучающихся по специальности

10.02.05 Обеспечение информационной безопасности автоматизированных систем

Ставрополь, 2020 г.

Настоящие методические указания составлены в соответствии с требованиями Федерального государственного образовательного стандарта среднего профессионального образования по специальности 10.02.05 Обеспечение информационной безопасности автоматизированных систем, и программой дисциплины «Криптографические средства защиты информации**»**

Рассмотрено на заседании методического объединения укрупненных групп специальностей 40.00.00 Юриспруденция Протокол № 1 от 25.08.2020 г.

Рекомендовано к использованию в учебном процессе Методическим советом СМК, протокол № 1 от 31.08.2020 г.

Содержание

[Практическое занятие №1 Введение в криптографическую защиту информации 4](#_Toc954188)

[Практическое занятие №2ОСНОВЫ КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ 12](#_Toc954189)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 3 ПОТОКОВОЕ ШИФРОВАНИЕ ДАННЫХ 18](#_Toc954190)

[Практическое занятие № 4 ИССЛЕДОВАНИЕ МЕТОДОВ ПОЛИАЛФАВИТНОЙ ПОДСТАНОВКИ 22](#_Toc954191)

[Практическое занятие № 5 ШИФРОВАНИЕ МЕТОДОМ ПЕРЕСТАНОВКИ 24](#_Toc954192)

[Практическое занятие № 6 ИССЛЕДОВАНИЕ ГАММИРОВАНИЯ ПРИ ШИФРОВАНИИ 26](#_Toc954193)

[Практическое занятие № 7 РАЗРАБОТКА И ИССЛЕДОВАНИЕ КРИПТОАЛГОРИТМА НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ СКРЕМБЛЕРА 32](#_Toc954194)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 8 АЛГОРИТМ БЛОЧНОГО ШИФРОВАНИЯ ДАННЫХ ГОСТ 28147-89 34](#_Toc954195)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 9 СИММЕТРИЧНОЕ ШИФРОВАНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ КРИПТОГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ MICROSOFT CRYPTOAPI И CRYPTOGRAPHY API: NEXT GENERATION 36](#_Toc954196)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 10 СИММЕТРИЧНОЕ И АСИММЕТРИЧНОЕ ШИФРОВАНИЕ ДАННЫХ СРЕДСТВАМИ КРИПТОГРАФИЧЕСКОГО ПАКЕТА OPENSSL 57](#_Toc954197)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 12 РЕАЛИЗАЦИЯ ЗАЩИЩЕННОЙ ПЕРЕДАЧИ ДАННЫХ ПО ПРОТОКОЛУ TLS СРЕДСТВАМИ КРИПТОГРАФИЧЕСКОГО ПАКЕТА OPENSSL 94](#_Toc954198)

[Практическое занятие № 13. Шифрование сообщений с помощью шифров замены криптографическими функциями Excel 103](#_Toc954199)

[Практическое занятие № 14. Исследование работы простейших алгоритмов шифрования в среде программирования Delphi 116](#_Toc954200)

[Практическое занятие № 15. Исследование функций гаммирования на языке программирования Java 119](#_Toc954201)

[Практическое занятие № 16. Исследование возможностей блочного шифрования на языке программирования Java 124](#_Toc954202)

[Практическое занятие №17. Исследование возможностей поточного шифрования на языке программирования Java 132](#_Toc954203)

[Практическое занятие № 18. Программная реализация шифров на языке программирования Java 134](#_Toc954204)

[Практическое занятие № 19. Исследование блочного алгоритма шифрования DES 140](#_Toc954205)

[Практическое занятие № 20. Исследование алгоритма шифрования RSA в MS 148](#_Toc954206)

[Практическое занятие №22Средства обеспечения безопасности ОС семейства Windows 154](#_Toc954207)

[Практическое занятие N23 Асимметричная криптография и электронная цифровая подпись на примере си- стемы GnuPG 174](#_Toc954208)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 24 Аутентификация пользователей Web-систем средствами технологии РНР 190](#_Toc954209)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 25Защита информации с помощью пароля 190](#_Toc954210)

[ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №26 191](#_Toc954211)

[Практическое занятие №27 Удостоверяющие центры на основе службы сертификации в операционной системе Windows 2003 Server 191](#_Toc954212)

[Практическое занятие № 28. Защита программ от несанкционированного использования с помощью USB-ключей и программного обеспечения производителя. 199](#_Toc954218)

[Практическое занятие № 29. Защита программ от несанкционированного использования с помощью USB-ключей и средств разработчика. 202](#_Toc954223)

[Практическое занятие № 31. Защита данных с помощью USB-ключей и программного обеспечения производителя. 205](#_Toc954230)

[Практическое занятие № 32. Защита данных с помощью USB-ключей и средств разработчика. 206](#_Toc954231)

**Практическое занятие №1 Введение в криптографическую защиту информации**

Введение

Появление новых информационных технологий и развитие мощных компьютерных систем хранения и обработки информации повысили уровни защиты информации и вызвали необходимость того, чтобы эффективность защиты информации росла вместе со сложностью архитектуры хранения данных. Постепенно защита информации становится обязательной: разрабатываются всевозможные документы по защите информации; формируются рекомендации; даже проводится ФЗ о защите информации, который рассматривает проблемы и задачи защиты информации, а также решает некоторые уникальные вопросы защиты информации.

Таким образом, угроза защиты информации сделала средства обеспечения информационной безопасности одной из обязательных характеристик информационной системы.

Под информационной безопасностью Российской Федерации (информационной системы) подразумевается техника защиты информации от преднамеренного или случайного несанкционированного доступа и нанесения тем самым вреда нормальному процессу документооборота и обмена данными в системе, а также хищения, модификации и уничтожения информации.

Другими словами вопросы защиты информации решаются для того, чтобы изолировать нормально функционирующую информационную систему от несанкционированных управляющих воздействий и доступа посторонних лиц или программ к данным с целью хищения.

Под фразой «угрозы безопасности информационных систем» понимаются реальные или потенциально возможные действия или события, которые способны исказить хранящиеся в информационной системе данные, уничтожить их или использовать в каких-либо целях, не предусмотренных регламентом заранее.

Для обеспечения защиты информации в настоящее время не существует единого технического приема или средства, однако общим в решении многих проблем безопасности является использование криптографии и криптоподобных преобразований информации. Защита конфиденциальной информации, основанная на криптографической защите информации, шифрует данные при помощи семейства обратимых преобразований, каждое из которых описывается параметром, именуемым «ключом» и порядком, определяющим очередность применения каждого преобразования.

 Защита информации в БД также является актуальной задачей как при единоличном использовании БД, так и при совместной работе пользователей с ней. Защита должна обеспечивать неизменность и целостность БД и содержащейся в ней информации, а также регламентировать права доступа к ней.
 При корпоративной работе группы пользователей с одной базой данных необходимо выбрать администратора, обслуживающего БД и обладающего соответствующими правами доступа. Права доступа пользователей устанавливаются администратором, который может включать и исключать пользователей, разбивать их на группы. Пользователи, входящие в состав определенной группы, обладают всеми предоставленными ей правами. Если личные права пользователя выше прав доступа группы, то личные права за ним сохраняются.

Для организации эффективных мероприятий по защите информации требуется не только разработка модели механизмов защиты информации и средства защиты информации в сети, но также и реализация системного подхода по обеспечению безопасности информационных систем – использование комплекса взаимосвязанных мер по защите информации, включающих в себя специальные технические и программные средства.

1. Цель работы

Исследование основных методов криптографической зашиты информации.

2. Краткие сведения из теории

Криптография – обеспечивает сокрытие смысла сообщения с помощью шифрования и открытия его расшифрованием, которые выполняются по специальным алгоритмам с помощью ключей.

Ключ – конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных, обеспечивающее выбор только одного варианта из всех возможных для данного алгоритма.

Криптоанализ – занимается вскрытием шифра без знания ключа (проверка устойчивости шифра).

Кодирование – (не относится к криптографии) – система условных обозначений, применяемых при передаче информации. Применяется для увеличения качества передачи информации, сжатия информации и для уменьшения стоимости хранения и передачи.

Криптографические преобразования имеют цель обеспечить недоступность информации для лиц, не имеющих ключа, и поддержание с требуемой надежностью обнаружения несанкционированных искажений.

Большинство средств защиты информации базируется на использовании криптографических шифров и процедур шифрования - расшифрования. В соответствии со стандартом ГОСТ 28147-89 под шифром понимают совокупность обратимых преобразований множества открытых данных на множество зашифрованных данных, задаваемых ключом и алгоритмом преобразования.

В криптографии используются следующие основные алгоритмы шифрования:

* алгоритм замены (подстановки) – символы шифруемого текста заменяются символами того же или другого алфавита в соответствии с заранее обусловленной схемой замены;
* алгоритм перестановки – символы шифруемого текста переставляются по определенному правилу в пределах некоторого блока этого текста;
* гаммирование – символы шифруемого текста складываются с символами некоторой случайной последовательности;
* аналитическое преобразование – преобразование шифруемого текста по некоторому аналитическому правилу (формуле).

Процессы шифрования и расшифрования осуществляются в рамках некоторой криптосистемы. Для симметричной криптосистемы характерно применение одного и того же ключа как при шифровании, так и при расшифровании сообщений. В асимметричных криптосистемах для зашифрования данных используется один (общедоступный) ключ, а для расшифрования – другой (секретный) ключ.

2.1. Симметричные криптосистемы

2.1.1.Шифры перестановки

В шифрах средних веков часто использовались таблицы, с помощью которых выполнялись простые процедуры шифрования, основанные на перестановке букв в сообщении. Ключом в данном случае является размеры таблицы. Например, сообщение “Неясное становится еще более непонятным” записывается в таблицу из 5 строк и 7 столбцов по столбцам.

Таблица 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Н | О | Н | С | Б | Н | Я |
| Е | Е | О | Я | О | Е | Т |
| Я | С | В | Е | Л | П | Н |
| С | Т | И | Щ | Е | О | Ы |
| Н | А | Т | Е | Е | Н | М |

Для получения шифрованного сообщения текст считывается по строкам и группируется по 5 букв:

НОНСБ НЯЕЕО ЯОЕТЯ СВЕЛП НСТИЩ ЕОЫНА ТЕЕНМ

Несколько большей стойкостью к раскрытию обладает метод одиночной перестановки по ключу. Он отличается от предыдущего тем, что столбцы таблицы переставляются по ключевому слову, фразе или набору чисел длиной в строку таблицы. Используя в качестве ключа слово - ЛУНАТИК, получим следующую таблицу

Таблица 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Л | У | Н | А | Т | И | К |  |  | А | И | К | Л | Н | Т | У |
| 4 | 7 | 5 | 1 | 6 | 2 | 3 |  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Н | О | Н | С | Б | Н | Я |  |  | С | Н | Я | Н | Н | Б | О |
| Е | Е | О | Я | О | Е | Т |  |  | Я | Е | Т | Е | О | О | Е |
| Я | С | В | Е | Л | П | Н |  |  | Е | П | Н | Я | В | Л | С |
| С | Т | И | Щ | Е | О | Ы |  |  | Щ | О | Ы | С | И | Е | Т |
| Н | А | Т | Е | Е | Н | М |  |  | Е | Н | М | Н | Т | Е | А |

 До перестановки После перестановки

В верхней строке левой таблицы записан ключ, а номера под буквами ключа определены в соответствии с естественным порядком соответствующих букв ключа в алфавите. Если в ключе встретились бы одинаковые буквы, они бы нумеровались слева направо. Получается шифровка: СНЯНН БОЯЕТ ЕООЕЕ ПНЯВЛ СЩОЫС ИЕТЕН МНТЕА. Для обеспечения дополнительной скрытности можно повторно шифровать сообщение, которое уже было зашифровано. Для этого размер второй таблицы подбирают так, чтобы длины ее строк и столбцов отличались от длин строк и столбцов первой таблицы. Лучше всего, если они будут взаимно простыми.

Кроме алгоритмов одиночных перестановок применяются алгоритмы двойных перестановок. Сначала в таблицу записывается текст сообщения, а потом поочередно переставляются столбцы, а затем строки. При расшифровке порядок перестановок был обратный. Пример данного метода шифрования показан в следующих таблицах:

Двойная перестановка столбцов и строк

Таблица 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 4 | 1 | 3 |   | 1 | 2 | 3 | 4 |   | 1 | 2 | 3 | 4 |
| 4 | П | Р | И | Е | 4 | И | П | Е | Р | 1 | А | З | Ю | Ж |
| 1 | З | Ж | А | Ю | 1 | А | 3 | Ю | Ж | 2 | Е | - | С | Ш |
| 2 | - | Ш | Е | С | 2 | Е | - | С | Ш | 3 | Г | Т | О | О |
| 3 | Т | О | Г | О | 3 | Г | Т | О | О | 4 | И | П | Е | Р |

В результате перестановки получена шифровка АЗЮЖЕ\_СШГТООИПЕР. Ключом к шифру служат номера столбцов 2413 и номера строк 4123 исходной таблицы.

Число вариантов двойной перестановки достаточно быстро возрастает с увеличением размера таблицы: для таблицы 3 х 3 их 36, для 4 х 4 их 576, а для 5\*5 их 14400.

В средние века для шифрования применялись и магические квадраты. Магическими квадратами называются квадратные таблицы с вписанными в их клетки последовательными натуральными числами, начиная с единицы, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и то же число. Для шифрования необходимо вписать исходный текст по приведенной в квадрате нумерации и затем переписать содержимое таблицы по строкам. В результате получается шифротекст, сформированный благодаря перестановке букв исходного сообщения.

Таблица 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| П | Р | И | Е | З | Ж | А | Ю | \_ | Ш | Е | С | Т | О | Г | О |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16 | 3 | 2 | 13 |   |   | О | И | Р | Т |
| 5 | 10 | 11 | 8 |   |   | З | Ш | Е | Ю |
| 9 | 6 | 7 | 12 |   |   | - | Ж | А | С |
| 4 | 15 | 14 | 1 |   |   | Е | Г | О | П |

Число магических квадратов очень резко возрастает с увеличением размера его сторон: для таблицы 3\*3 таких квадратов -1; для таблицы 4\*4 – 880; а для таблицы 5\*5-250000.

2.2. Шифры простой замены

Система шифрования Цезаря - частный случай шифра простой замены. Метод основан на замене каждой буквы сообщения на другую букву того же алфавита, путем смещения от исходной буквы на K букв.

Известная фраза Юлия Цезаря VENI VINI VICI – пришел, увидел, победил, зашифрованная с помощью данного метода, преобразуется в SBKF SFAF SFZF (при смещении на 4 символа).

Греческим писателем Полибием за 100 лет до н.э. был изобретен так называемый полибианский квадрат размером 5\*5, заполненный алфавитом в случайном порядке. Греческий алфавит имеет 24 буквы, а 25-м символом является пробел. Для шифрования на квадрате находили букву текста и записывали в шифротекст букву, расположенную ниже ее в том же столбце. Если буква оказывалась в нижней строке таблицы, то брали верхнюю букву из того же столбца.

2.3. Шифры сложной замены

Шифр Гронсфельда состоит в модификации шифра Цезаря числовым ключом. Для этого под буквами сообщения записывают цифры числового ключа. Если ключ короче сообщения, то его запись циклически повторяют. Шифротекст получают примерно так же, как в шифре Цезаря, но отсчитывают не третью букву по алфавиту (как в шифре Цезаря), а ту, которая смещена по алфавиту на соответствующую цифру ключа.

Пусть в качестве ключа используется группа из трех цифр – 314, тогда

Сообщение СОВЕРШЕННО СЕКРЕТНО

Ключ 3143143143143143143

Шифровка ФПЖИСЬИОССАХИЛФИУСС

В шифрах многоалфавитной замены для шифрования каждого символа исходного сообщения применяется свой шифр простой замены (свой алфавит).

Таблица 5

|  |  |
| --- | --- |
|   | АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_ |
| А | АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_ |
| Б | \_АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ |
| В | Я\_АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮ |
| Г | ЮЯ\_АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭ |
| . | ………… |
| Я | ВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_АБ |
| \_ | БВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_А |

Каждая строка в этой таблице соответствует одному шифру замены аналогично шифру Цезаря для алфавита, дополненного пробелом. При шифровании сообщения его выписывают в строку, а под ним ключ. Если ключ оказался короче сообщения, то его циклически повторяют. Шифротекст получают, находя символ в колонке таблицы по букве текста и строке, соответствующей букве ключа. Например, используя ключ АГАВА, из сообщения ПРИЕЗЖАЮ ШЕСТОГО получаем следующую шифровку:

Таблица 6

|  |  |
| --- | --- |
| Сообщение | ПРИЕЗЖАЮ\_ШЕСТОГО |
| Ключ | АГАВААГАВААГАВАА |
| Шифровка | ПНИГЗЖЮЮЮШЕОТМГО |

В компьютере такая операция соответствует сложению кодов ASCII символов сообщения и ключа по модулю 256.

2.4. Гаммирование

Процесс зашифрования заключается в генерации гаммы шифра и наложении этой гаммы на исходный открытый текст. Перед шифрованием открытые данные разбиваются на блоки Т(0)i одинаковой длины (по 64 бита). Гамма шифра вырабатывается в виде последовательности блоков Г(ш)i аналогичной длины (Т(ш)i=Г(ш)i+Т(0)i, где + - побитовое сложение, i =1-m).

Процесс расшифрования сводится к повторной генерации шифра текста и наложение этой гаммы на зашифрованные данные T(0)i=Г(ш)i+Т(ш)i.

Исходное сообщение из букв русского алфавита преобразуется в числовое сообщение заменой каждой его буквы числом по следующей таблице:

Таблица 7

|  |
| --- |
| Числовая замена букв |
| А | Б | В | Г | Д | Е | Ж | З | И | К | Л | М | Н  | О | П |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 |
| Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |

Ωi+1= [(Ai + C1 – 1) mod 30] + 1

Исходное сообщение ОТДУШКА. Для шифрования числового сообщения используется шифрующий отрезок последовательности А1, А2, ... подходящей длины, начинающийся с А100:

Таблица 8

|  |
| --- |
| Остатки при делении на 30 |
| А100 | А101 | А102 | А103 | А104 | А105 | А106 |
| 1 | 5 | 6 | 17 | 18 | 19 | 3 |

При шифровании каждое число числового сообщения складывается с соответствующим числом шифрующего отрезка. Затем вычисляется остаток от деления полученной суммы на 30, который по данной таблице заменяется буквой.

Таблица 9

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Исходное сообщение | О | Т | Д | У | Ш | К | А |
| Числовое исходное сообщение | 13 | 17 | 4 | 18 | 23 | 9 | 0 |
| Шифрующий отрезок | 1 | 5 | 6 | 17 | 8 | 19 | 3 |
| Числовое шифрованное сообщение | 14 | 23 | 10 | 5 | 1 | 28 | 3 |
| Шифрованное сообщение | П | Ш | Л | Е | Б | Ю | Г |

2.5. Асимметричные криптосистемы

2.5.1. Схема шифрования Эль Гамаля

Алгоритм шифрования Эль Гамаля основан на применении больших чисел для генерации открытого и закрытого ключа, криптостойкость же обусловлена сложностью вычисления дискретных логарифмов.

Последовательность действий пользователя:

1. Получатель сообщения выбирает большое, простое число P и большое целое число G, причем P > G.
2. Получатель выбирает секретный ключ - случайное целое число X < P.
3. Вычисляется открытый ключ Y= G x mod P.
4. Получатель выбирает случайное целое число K, 1< K< P-1, такое, что числа К и (Р-1) являются взаимно простыми.
5. Шифрование сообщения (M): a= GK mod P, b=Y K M mod P, где пара чисел (a,b) является шифротекстом.
6. Надо придумать сообщение М, зашифровать его, а потом с помощью секретного ключа Х расшифровать сообщение.

Шифрование

* 1. Допустим что нужно зашифровать сообщение .
	2. Произведем генерацию ключей :
		1. пусть . Выберем - случайное целое число такое,что .
		2. Вычислим .
		3. Итак , открытым является тройка ,а закрытым ключом является число .
	3. Выбираем случайное целое число такое, что 1 < k < (p − 1). Пусть .
	4. Вычисляем число .
	5. Вычисляем число .
	6. Полученная пара является шифротекстом.

Расшифрование

* + 1. Необходимо получить сообщение по известному шифротексту и закрытому ключу .
		2. Вычисляем M по формуле : 
		3. Получили исходное сообщение .

2.5.2. Криптосистема шифрования данных RSA

Предложена в 1978 году авторами Rivest, Shamir и Aldeman и основана на трудности разложения больших целых чисел на простые сомножители.

Последовательность действий пользователя:

1. Получатель выбирает 2 больших простых целых числа p и q, на основе которых вычисляет N=pq; M=(p-1)(q-1).
2. Получатель выбирает целое случайное число d, которое является взаимопростым со значением М, и вычисляет значение е из условия ed=1/(mod M).
3. d и N публикуются как открытый ключ, е и М являются закрытым ключом.
4. Если S –сообщение и его длина: 1<Len(S)<N, то зашифровать этот текст можно как S’=Sd(mod N), то есть шифруется открытым ключом.
5. Получатель расшифровывает с помощью закрытого ключа: S=S’e(mod N).
6. Представить шифруемое сообщение как последовательность целых чисел в диапазоне 0…32. Пусть буква А изображается числом 1, буква Б- числом 2, буква В- числом 3 и т.д. Тогда сообщение будет принимать вид числовой последовательности, которое зашифровывается с помощью открытого ключа. Надо расшифровать его с помощью закрытого ключа.

Зашифруем сообщение «САВ». Для простоты будем использовать маленькие числа (на практике применяются гораздо большие).

* 1. Выберем p=3 и q=11.
	2. Определим n=3\*11=33.
	3. Найдем (p-1)(q-1)=20. Следовательно, в качестве d, взаимнопростое с 20, например, d=3.
	4. Выберем число е. В качестве такого числа может быть взято любое число, для которого удовлетворяет соотношение (е\*3)(mod20)=1, например 7.
	5. Представим шифруемое сообщение как последовательность целых чисел с помощью отображения: А1, В2, С3. Тогда сообщение принимает вид (3,1,2). Зашифруем сообщение с помощью ключа {7,33}.
	ШТ1 = (37) (mod 33) = 2187 (mod 33) = 9,
	ШТ2 = (17) (mod 33) = 1 (mod 33) = 1,
	ШТ3 = (27) (mod 33) = 128 (mod 33) = 29.
	6. Расшифруем полученное зашифрованное сообщение (9,1,29) на основе закрытого ключа {3,33}:
	ИТ1 = (93) (mod 33) = 729 (mod 33) = 3,
	ИТ2 = (13) (mod 33) = 1 (mod 33) = 1,
	ИТ3 = (293) (mod 33) = 24389 (mod 33) = 2.

Итак в реальных системах алгоритм RSA реализуется следующим образом: каждый пользователь выбирает два больших числа, и в соответствии с описанным выше алгоритмом выбирает два простых числа e и d. Как результат умножения первых двух чисел (p и q) устанавливается n.

{e,n} образует открытый ключ, а {d,n} – закрытый (хотя можно взять и наоборот).

Открытый ключ публикуется и доступен каждому, кто желает послать владельцу ключа сообщение, которое зашифровывается указанным алгоритмом. После шифрования, сообщение невозможно раскрыть с помощью открытого ключа. Владелец же закрытого ключа без труда может расшифровать принятое сообщение.

3. Задание к работе

 Зашифровать любыми пятью методами свои данные: Фамилию, Имя, Отчество, любимую фразу.

 Контрольные вопросы

1. Объяснить цель и задачи криптографии.
2. Шифры одиночной перестановки и перестановки по ключевому слову. Шифр Гронфельда.
3. Шифры двойной перестановки. Шифрование с помощью магического квадрата.
4. Шифр многоалфавитной замены и алгоритм его реализации.
5. Пояснить алгоритм шифрации двойным квадратом. Шифр Enigma.
6. Пояснить алгоритм шифрования DES.
7. Пояснить алгоритм шифрования ГОСТ 28147-89.
8. Пояснить алгоритм шифрования RSA.
9. Пояснить алгоритм шифрования Эль Гамаля.
10. Каковы задачи и алгоритмы электронной подписи?
11. Какие задачи распределения ключей?

**Практическое занятие №2 ОСНОВЫ КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ**

Теоретическая часть

Криптография – обеспечивает сокрытие смысла сообщения с помощью шифрования и открытия его расшифрованием, которые выполняются по специальным алгоритмам с помощью ключей.

Ключ – конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных, обеспечивающее выбор только одного варианта из всех возможных для данного алгоритма.

Криптоанализ – занимается вскрытием шифра без знания ключа (проверка устойчивости шифра).

Кодирование – (не относится к криптографии) – система условных обозначений, применяемых при передаче информации. Применяется для увеличения качества передачи информации, сжатия информации и для уменьшения стоимости хранения и передачи.

Криптографические преобразования имеют цель обеспечить недоступность информации для лиц, не имеющих ключа, и поддержание с требуемой надежностью обнаружения несанкционированных искажений.

Большинство средств защиты информации базируется на использовании криптографических шифров и процедур шифрования - расшифрования. В соответствии со стандартом ГОСТ 28147-89 под шифром понимают совокупность обратимых преобразований множества открытых данных на множество зашифрованных данных, задаваемых ключом и алгоритмом преобразования.

В криптографии используются следующие основные алгоритмы шифрования:

• алгоритм замены (подстановки) – символы шифруемого текста заменяются символами того же или другого алфавита в соответствии с заранее обусловленной схемой замены;

• алгоритм перестановки – символы шифруемого текста переставляются по определенному правилу в пределах некоторого блока этого текста;

• гаммирование – символы шифруемого текста складываются с символами некоторой случайной последовательности;

• аналитическое преобразование – преобразование шифруемого текста по некоторому аналитическому правилу (формуле).

Процессы шифрования и расшифрования осуществляются в рамках некоторой криптосистемы. Для симметричной криптосистемы характерно применение одного и того же ключа как при шифровании, так и при расшифровании сообщений. В асимметричных криптосистемах для зашифрования данных используется один (общедоступный) ключ, а для расшифрования – другой (секретный) ключ.

Практическая часть

1. Шифры перестановки

В шифрах средних веков часто использовались таблицы, с помощью которых выполнялись простые процедуры шифрования, основанные на перестановке букв в сообщении. Ключом в данном случае является размеры таблицы. Например, сообщение «ЛАМАНТИН КУПИЛ ПЛАЩ ДОРОГОЙ» записывается в таблицу из 4 строк и 6 столбцов по столбцам.

Таблица 1

Л Н К Л Щ О

А Т У П Д Г

М И П Л О О

А Н И А Р Й

Для получения шифрованного сообщения текст считывается по строкам и группируется по 4 буквы: ЛНКЛ ЩОАТ УПДГ МИПЛ ООАН ИАРЙ. Метод одиночной перестановки по ключу.

Он отличается от предыдущего тем, что столбцы таблицы переставляются по ключевому слову, фразе или набору чисел длиной в строку таблицы. Используя в качестве ключа слово, ПАЛАТА, получим следующую таблицу:

Таблица 2

П А Л А Т А А А А Л П Т

5 1 4 2 6 3 1 2 3 4 5 6

Л Н К Л Щ О Н Л О К Л Щ

А Т У П Д Г Т П Г У А Д

М И П Л О О И Л О П М О

А Н И А Р Й Н А Й И А Р

До перестановки После перестановки

В верхней строке левой таблицы записан ключ, а номера под буквами ключа определены в соответствии с естественным порядком соответствующих букв ключа в алфавите. Если в ключе встретились бы одинаковые буквы, они бы нумеровались слева направо.

Получается шифровка: НЛОК ЛЩТП ГУАД ИЛОП МОНА ЙИАР.

Алгоритмы двойных перестановок

Сначала в таблицу записывается текст сообщения, а потом поочередно переставляются столбцы, а затем строки. При расшифровке порядок перестановок был обратный. Пример данного метода шифрования показан в следующих таблицах:

Таблица 3

 4 2 1 3 1 2 3 4 1 2 3 4

3 С Л О В 3 О Л В С 1 О В Р \_

2 О \_ Н Е 2 Н \_ Е О 2 Н \_ Е О

1 \_ В О Р 1 О В Р \_ 3 О Л В С

4 О Б Е Й 4 Е Б Й О 4 Е Е Й О

В результате двойной перестановки столбцов и строк получена шифровка ОВР\_Н\_ЕООЛВСЕЕЙО. Ключом к шифру служат номера столбцов 3214 и номера строк 4213 исходной таблицы.

Магические квадраты

Магическими квадратами называются квадратные таблицы с вписанными в их клетки последовательными натуральными числами, начиная с единицы, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и то же число. Для шифрования необходимо вписать исходный текст по приведенной в квадрате нумерации и затем переписать содержимое таблицы по строкам. В результате получается шифротекст, сформированный благодаря перестановке букв исходного сообщения:

Таблица 4

С Л О В О \_ Н Е \_ В О Р О Б Е Й

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

16 3 2 13 Й О Л О

5 10 11 8 О В О Е

9 6 7 12 \_ \_ Е Р

4 15 14 1 В Е Б С

2. Шифры простой замены

Система шифрования Цезаря - частный случай шифра простой замены. Метод основан на замене каждой буквы сообщения на другую букву того же алфавита, путем смещения от исходной буквы на K букв.

ШИФР ЦЕЗАРЯ (смещение на 2 символа)

ХЖТО ФГЁЮОЭ Таблица 5

 АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_

А АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_

Б \_АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ

В Я\_АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮ

Г ЮЯ\_АБВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭ

. …………

Я ВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_АБ

\_ БВГДЕЁЖЗИКЛМНОПРСТУФХЧШЩЪЫЬЭЮЯ\_А

3. Шифры сложной замены

Шифр Гронсфельда состоит в модификации шифра Цезаря числовым ключом. Для этого под буквами сообщения записывают цифры числового ключа. Если ключ короче сообщения, то его запись циклически повторяют. Шифротекст получают примерно также, как в шифре Цезаря, но отсчитывают не третью букву по алфавиту (как в шифре Цезаря), а ту, которая смещена по алфавиту на соответствующую цифру ключа. Каждая строка в этой таблице соответствует одному шифру замены аналогично шифру Цезаря для алфавита, дополненного пробелом. При шифровании сообщения его выписывают в строку, а под ним ключ. Если ключ оказался короче сообщения, то его циклически повторяют. Шифротекст получают, находя символ в колонке таблицы по букве текста и строке, соответствующей букве ключа.

Пусть в качестве ключа используется группа из трех цифр – 132, тогда

Сообщение СЛОВО НЕ ВОРОБЕЙ

Ключ 13213213213213213213213213

Шифровка РЗМВМЛМАМПЛ\_ДЖ

Можно также использовать ключ, состоящий из букв, например, АБВА:

Сообщение ВСЕМУ\_СВОЕ\_ВРЕМЯ

Ключ АБВААБВААБВААБВА

Шифровка ВРГМУЯПВОДЮВРДКЯ

4. Гаммирование

Процесс зашифрования заключается в генерации гаммы шифра и наложении этой гаммы на исходный открытый текст. Перед шифрованием открытые данные разбиваются на блоки Т(0)i одинаковой длины (по 64 бита). Гамма шифра вырабатывается в виде последовательности блоков Г(ш)i аналогичной длины (Т(ш)i = Г(ш)i + Т(0)i, где + - побитовое сложение, i =1-m).

Процесс расшифрования сводится к повторной генерации шифра текста и наложение этой гаммы на зашифрованные данные T(0)i = Г(ш)i + Т (ш)i.

Пронумеруем буквы русского алфавита и переведем номера соответствующие буквам в двоичную систему исчисления.

Таблица 6

Числовая замена букв

А Б В Г Д Е Ж З И К Л М Н О П

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14

Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Э Ю Я

15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Для шифрования числового сообщения используется шифрующий отрезок последовательности подходящей длины. При шифровании каждое число числового сообщения складывается с соответствующим числом шифрующего отрезка. Затем вычисляется остаток от деления полученной суммы на 30, который по данной таблице заменяется буквой.

Теперь запишем слово “ЛОЖКА”, заменив буквы соответствующими им двоичными цифрами.

Таблица 7

Исходное сообщение Л О Ж К А

Числовое исходное сообщение 10 13 6 9 0

Шифрующий отрезок 3 1 15 10 6

Числовое шифрованное сообщение 13 14 21 19 6

Шифрованное сообщение О П Ц Ф Ж

5. Асимметричные криптосистемы

Схема шифрования Эль Гамаля

Алгоритм шифрования Эль Гамаля основан на применении больших чисел для генерации открытого и закрытого ключа, криптостойкость же обусловлена сложностью вычисления дискретных логарифмов.

Последовательность действий пользователя:

1. Получатель сообщения выбирает два больших числа P и G, причем P > G. P = 15, G = 4.

2. Получатель выбирает секретный ключ - случайное целое число X < P.

Х = 8.

3. Вычисляется открытый ключ Y = GX mod P.

Y = 48 mod15 = 1.

4. Получатель выбирает целое число K, 1< K< P-1, такое, что числа К и (Р-1) являются взаимно простыми.

K = 11.

5. Шифрование сообщения (M): a= GK mod P, b=Y K M mod P, где пара чисел (a,b) является шифротекстом.

M = 7;

a = GK mod P = 411 mod15 = 4;

b =YK M mod P = 111 ·7 mod15 = 7.

Пара чисел (4, 7) является шифротекстом.

Расшифровка:

M = b/aX mod P = 7/48 mod15 = 7.

Пояснение:

M = b/aX mod P = YK ·M/GKX mod P = GKX mod P·M/GKX mod P = M.

Криптосистема шифрования данных RSA

Предложена в 1978 году авторами Rivest, Shamir и Aldeman и основана на трудности разложения больших целых чисел на простые сомножители.

Последовательность действий пользователя:

1. Получатель выбирает 2 больших простых целых числа p и q, на основе которых вычисляет

N = pq; M = (p-1)(q-1).

p = 7, q = 23, N = p·q = 161;

M = (p-1)(q-1) = 132.

2. Получатель выбирает целое случайное число d, которое является взаимопростым со значением М, и вычисляет значение е из условия ed = 1/(mod M).

d = 7;

ed = 1/(mod M);

e·7 mod 132 = 1;

e = 19.

3. d и N публикуются как открытый ключ, е и М являются закрытым ключом.

4. Если S – сообщение и его длина: 1 < Len(S) < N, то зашифровать этот текст можно как S’=Sd(mod N), то есть шифруется открытым ключом.

S = 15;

S’ = Sd(mod N) = 157 mod 161 = 57.

5. Получатель расшифровывает с помощью закрытого ключа: S=S’e(mod N).

S = S’e(mod N) = 5719 mod 161 = 15.

ключ алгоритм магический квадрат

Вывод: В ходе лабораторной работы были изучены основные методы криптографической зашиты информации, также была произведена шифровка всеми методами криптографии.

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 3 ПОТОКОВОЕ ШИФРОВАНИЕ ДАННЫХ**

Цель работы: получить навыки в создании программной реализации алгоритма потокового шифрования на базе регистра сдвига с линейной обратной связью.

ОСНОВНЫЕ ПОНЯТИЯ

Потоковые шифры преобразуют открытый текст в шифртекст по одному биту за операцию. Простейшая реализация потокового шифра показана на рис. 1.1. Генератор потока ключей (иногда называемый генератором с бегущим ключом или генератор гаммы) выдает псевдослучайный поток битов: k1, k2, k3, ..., ki. Этот поток ключей (гамма шифра) и поток битов открытого текста, p1, p2, p3, ..., pi, подвергаются операции «исключающее или» (XOR), и в результате получается поток битов шифртекста

ci = pi  ki

При расшифровании операция XOR выполняется над битами шифртекста и тем же самым потоком ключей для восстановления битов открытого текста

pi = ci  ki

Так как

pi  ki  ki = pi,

данная схема работает правильно.

Безопасность системы полностью зависит от свойств генератора потока ключей. Если генератор потока ключей выдает бесконечную строку нулей, шифртекст будет совпадать с открытым текстом, и вся операция будет бессмысленна. Чем ближе выход генератора потока ключей к случайному, тем больше времени потребуется криптоаналитику, чтобы взломать шифр.

Pi Pi

текст

Зашифрование

текст

Расшифрование

Большинство потоковых шифров основано на регистрах сдвига с обратной связью, которые состоят из двух частей: собственно сдвигового регистра и функции обратной связи (см. рис. 1.2).

bn-1 bn -2 .... b3 b2 b1 b0

Сдвиговый регистр представляет собой последовательность битов. Количество битов определяется длиной регистра. Если длина равна n битам, то регистр называется n-битовым регистром сдвига. Всякий раз, когда нужно извлечь бит, все биты регистра сдвига сдвигаются вправо на 1 позицию. Новый крайний левый бит является функцией всех остальных битов регистра. На выходе регистра сдвига оказывается один, обычно младший значащий, бит. Периодом регистра сдвига называется длина получаемой последовательности до начала ее повторения.

К простейшему типу регистра сдвига с обратной связью относится регистр сдвига с линейной обратной связью (РСЛОС). Обратная связь представляет собой просто операцию XOR над некоторыми битами регистра. Перечень этих битов называется отводной

последовательностью. n-битовый РСЛОС может находиться в одном из 2n1 внутренних состояний (так как заполнение регистра нулями бесполезно).

Для представления отводной последовательности принято использовать двоичные многочлены вида xn + xa + xb + xc + xd +…+ x + 1, где n – длина регистра, a, b, c, d и т.д. – номера битов, задающие последовательность отводов. Например:

x32 + x7 +x5 + x3 + x2 + x + 1

Кроме того, данный многочлен можно представить в виде (32, 7, 5, 3, 2, 1, 0). И эта, и предыдущая запись означает, что для данного 32-битового регистра сдвига новый бит генерируется с помощью операции XOR над седьмым, пятым, третьим, вторым, первым и нулевым битом (см. рис. 1.3).

Выходной бит

Рис. 1.3. 32-битовый РСЛОС

Для того чтобы конкретный РСЛОС имел максимальный период (т.е. циклически проходил через все 2n1 внутренних состояний), многочлен, образованный из отводной последовательности, должен быть примитивным (неприводимым) – т.е. не раскладываться на произведение двоичных многочленов меньшей степени. Некоторые неприводимые двоичные многочлены приведены в табл. 1.1.

Перед запуском РСЛОС при зашифровании или расшифровании, регистр необходимо инициализировать одним и тем же значением.

Таблица 1.1

Некоторые неприводимые двоичные многочлены

(1, 0)

(2, 1, 0)

(3, 1, 0)

(4, 1, 0)

(5, 2, 0)

(6, 1, 0) (19, 5, 2, 1, 0)

(20, 3, 0)

(21, 2, 0)

(22, 1, 0)

(23, 5, 0)

(24, 4, 3, 1, 0) (36, 11, 0)

(36, 6, 5, 4, 2, 1, 0)

(37, 6, 4, 1, 0)

(37, 5, 4, 3, 2, 1, 0) (52, 3, 0)

(53, 6, 2, 1, 0)

(54, 8, 6, 3, 0)

(54, 6, 5, 4, 3, 2, 0)

(55, 24, 0)

(55, 6, 2, 1, 0)

Окончание табл. 1.1

(7, 1, 0)

(7, 3, 0)

(8, 4, 3, 2, 0)

(9, 4, 0)

(10, 3, 0)

(11, 2, 0)

(12, 6, 4, 1, 0)

(13, 4, 3, 1, 0)

(14, 5, 3, 1, 0)

(16, 5, 3.2, 0)

(17, 3, 0) (25, 3, 0)

(26, 6, 2, 1, 0)

(27, 5, 2, 1, 0)

(28, 3, 0)

(29, 2, 0)

(30, 6, 4, 1.0)

(31, 3, 0)

(31, 6, 0)

(31, 7, 0)

(32, 7, 6, 2, 0)

(32, 7, 5, 3, 2, 1, 0) (38, 6, 5, 1, 0)

(39, 4, 0)

(40, 5, 4, 3, 0)

(41, 3, 0)

(42, 7, 4, 3, 0)

(42, 5, 4, 3, 2, 1,

0)

(43, 6, 4, 3, 0)

(44, 6, 5, 2, 0)

(45, 4, 3, 1, 0)

(46, 8, 7, 6, 0)

(48, 9, 7, 4, 0)

(48, 7, 5, 4, 2, 1,

0) (56, 7, 4, 2, 0)

(57, 7, 0)

(57, 5, 3, 2, 0)

(58, 19.0)

(58, 6, 5, 1, 0)

(59, 7, 4, 2, 0)

(59, 6, 5, 4, 3, 1, 0)

(60, 1, 0)

(61, 5, 2, 1, 0)

(63, 1, 0)

(64, 4, 3, 1, 0)

СОДЕРЖАНИЕ РАБОТЫ

Разработать на выбранном языке программирования консольное или оконное приложение, реализующее описанный выше алгоритм для шифрования содержимого текстового или двоичного файла. Программа должна запрашивать имя входного и выходного файлов, представление образующего многочлена и инициализирующее значение. Разрядность РСЛОС должна быть меньше или равной максимальной разрядности стандартных целочисленных типов данных (64 бит).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Принципы работы потоковых шифров.

2. Какие операции используются при реализации потоковых шифров?

3. Что такое гамма шифра?

4. Что представляет собой регистр сдвига с линейной обратной связью?

5. Что такое отводная последовательность, и в какой форме ее можно представить?

6. Что такое период регистра сдвига?

7. Какие условия должны соблюдаться для того, чтобы РСЛОС имел максимальный период?

**Практическое занятие № 4 ИССЛЕДОВАНИЕ МЕТОДОВ ПОЛИАЛФАВИТНОЙ ПОДСТАНОВКИ**

Цель работы

Изучение принципов построения моноалфавитных и полиалфавит- ных шифров замены. Исследование свойств подстановочных шифров.

Теоретическая часть

Моноалфавитные шифры замены имели существенный недостаток – они легко поддавались частотному криптоанализу. Возникла потребность в разработке более устойчивых методов шифрования. Так на смену моноал- фавитным шифрам пришли шифры полиалфавитные.

Метод Виженера относится к числу полиалфавитных шифров заме- ны. Берется небольшое целое число m и алфавит после каждой символьной подстановки сдвигается на m символов.

Например, если ключом будет слово мышь (смотри левую вертикаль- ную колонку символов), тогда m = 4, при этом получаем следующую таб- лицу:

абвгдеёжзийклмнопрстуфхцчшщъыьэюя

1 мнопрстуфхцчшщъыьэюяабвгдеёжзийкл

2 ыьэюяабвгдеёжзийклмнопрстуфхцчшщъ

3 шщъыьэюяабвгдеёжзийклмнопрстуфхцч

4 ьэюяабвгдеёжзийклмнопрстуфхцчшщъы

Исходный текст разбивается на группы по m символов (в рассмот- ренном случае – по 4). Для каждой группы первый символ заменяется со- ответствующей буквой из первого алфавита, второй – из второго и т.д. На- пример, фраза «от улыбки каждый день светлей» будет преобразована сле- дующим образом:

отул ыбки кажд ыйде ньсв етле й ынлз зьге чыяа зеьб ъчйю сндб ц

Алфавит не ограничивается только лишь буквами, в него можно до- бавить и другие символы – пробел, цифры, знаки препинания. Такая моди- фикация позволит избежать двусмысленности при чтении текста после расшифровки на приёмной стороне (например, проблема простановки за- пятой во фразе «казнить нельзя помиловать»).

Ход работы

Получите индивидуальное задание у преподавателя. Реализуйте про- граммный модуль в соответствии с полученным заданием. При реализации необходимо учесть следующие моменты:

1) предусмотреть возможность задания пользователем своего абсо- лютно произвольного алфавита, состоящего из любого набора символов, расположенных в любом порядке;

2) для удобства тестирования и взаимодействия с другими модулями реализовать файловый ввод исходных данных и файловый вывод результа- та криптографического преобразования.

После реализации программного модуля выполните статистический анализ текста до криптографического преобразования и после него.

Варианты заданий

1. Модуль для шифрования текста по алгоритму Виженера, ключ –

слово или фраза.

2. Модуль для расшифровывания текста по алгоритму Виженера, ключ – слово или фраза.

3. Модуль для шифрования текста по алгоритму Виженера, ключ –

числовая последовательность.

4. Модуль для расшифровывания текста по алгоритму Виженера, ключ – числовая последовательность.

5. Модуль для шифрования текста по алгоритму Гронсфельда, ключ –

слово (до 10 символов).

6. Модуль для расшифровывания текста по алгоритму Гронсфельда, ключ – слово (до 10 символов).

7. Модуль для шифрования текста по алгоритму Гронсфельда, ключ –

числовая последовательность.

8. Модуль для расшифровывания текста по алгоритму Гронсфельда, ключ – числовая последовательность.

Содержание отчёта

1. Цель работы.

2. Задание.

3. Анализ задания.

4. Алгоритм преобразования.

5. Программа на алгоритмическом языке.

6. Тестовые запуски и статистический анализ.

7. Выводы по работе.

Контрольные вопросы

1. Какой шифр будет реализовывать алгоритм Виженера при ис- пользовании ключа, состоящего из одного символа?

2. В чём состоит принципиальная разница моноалфавитных и поли- алфавитных шифров замены?

3. Возможно ли применение статистических методов криптоанализа к полиалфавитным шифрам?

**Практическое занятие № 5 ШИФРОВАНИЕ МЕТОДОМ ПЕРЕСТАНОВКИ**

Цель работы

Изучение принципов построения шифров перестановки. Исследова- ние свойств перестановочных шифров.

Теоретическая часть

Метод заключается в том, что символы шифруемого текста пере- ставляются по определенным правилам внутри шифруемого блока, при этом сами символы не изменяются.

Самая простая перестановка – написать исходный текст задом напе- ред и одновременно разбить шифрограмму на пятерки букв. Например:

исходный текст: пусть будет так, как мы хотели подготовленный текст: пусть будет такка кмыхо тели зашифрованный текст: илето хымка ккатт едубь тсуп

В последней группе (пятерке) не хватает одной буквы. Значит, преж- де чем зашифровать исходное выражение, следует его дополнить незнача- щей буквой, например О, до числа, кратного пяти:

пусть будет такка кмыхо телио

Тогда шифрограмма будет выглядеть следующим образом: оилет охымк аккат тедуб ьтсуп

Другой метод заключается в том, что исходный текст записывали в несколько строк, например по пятнадцать букв в каждой (с заполнением последней строки незначащими буквами):

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

п у с т ь б у д е т т а к к а

к м ы х о т е л и к л м н о п

После этого вертикальные столбцы по порядку пишутся в строку с разбивкой на пятерки букв:

пкумс ытхьо бтуед леитк тламк нкоап

Если строки укоротить, а количество строк увеличить, то получится

прямоугольник – решетка, в которую записывается исходный текст.

Например:

1 2 3 4 5 6

п у с т ь б

у д е т т а

к к а к м ы

х о т е л и

а б в г д е

м л к и з ж

Если шифровать по диагоналям сверху вниз с левого верхнего угла, то получим:

п уу сдк текх ьтаоа бтктбм амевл ылгк иди ез ж

Третий вид данного шифра: перестановки с ключом. Необходимо знать ключ, например «радиатор». В соответствии с расположением букв в алфавите буква А получает номер 1, вторая буква А – 2, следующая по ал- фавиту буква Д – 3, потом И – 4, О – 5, первая буква Р – 6, вторая Р – 7 и буква Т – 8.

В результате получается:

Р А Д И А Т О Р

6 1 3 4 2 8 5 7

п у с т ь б у д

е т т а к к а к

м ы х о т е л и

о

Записывая столбики в соответствии с номерами букв ключа, получим: уты ькт стх тао уал пемо дки бке

Модификацией последнего метода является использование двух ключей: одного – для перестановки столбцов, а другого – для перестановки строк. Такой метод называется двойной перестановкой.

Ещё одним методом перестановки является использование решёток Кардано (шифр «Поворотная решётка»).

Ход работы

Получите индивидуальное задание у преподавателя. Реализуйте про- граммный модуль в соответствии с полученным заданием. При реализации необходимо учесть следующие моменты:

1) предусмотреть возможность задания пользователем параметров шифра;

2) предусмотреть визуализацию всех пользовательских настроек;

3) для удобства тестирования и взаимодействия с другими модулями реализовать файловый ввод исходных данных и файловый вывод результа- та криптографического преобразования.

После реализации программного модуля выполните статистический анализ текста по биграммам до криптографического преобразования и по- сле него.

Варианты заданий

1. Модуль для шифрования текста с помощью решёток Кардано.

2. Модуль для расшифровывания текста с помощью решёток Кардано.

3. Модуль для шифрования текста по алгоритму двойной переста- новки.

4. Модуль для расшифровывания текста по алгоритму двойной пере- становки.

Содержание отчёта

1. Цель работы.

2. Задание.

3. Анализ задания.

4. Алгоритм преобразования.

5. Программа на алгоритмическом языке.

6. Тестовые запуски и статистический анализ.

7. Выводы по работе.

Контрольные вопросы

1. Оцените количество возможных простых перестановок текста, со- стоящего из пяти символов? Из десяти символов? Из n символов?

2. Чем принципиально отличаются шифры перестановки от шифров замены?

3. Возможно ли применение статистических методов криптоанализа к перестановочным шифрам?

**Практическое занятие № 6 ИССЛЕДОВАНИЕ ГАММИРОВАНИЯ ПРИ ШИФРОВАНИИ**

Цель работы

Исследование методов генерации псевдослучайных последователь- ностей. Исследование гаммирования при шифровании данных.

Теоретическая часть

Гаммирование – это процесс наложения гаммы шифра на открытые данные по определенному закону.

Гамма шифра – псевдослучайная последовательность чисел, выраба- тываемая по заданному алгоритму для зашифровывания открытых данных и расшифровывания зашифрованных данных.

Суть метода заключается в том, что символы шифруемого текста по- следовательно складываются с символами некоторой специальной после- довательности, которая называется гаммой.

Процедуру наложения гаммы можно реализовать двумя способами.

1. Гаммирование по модулю К. Символы исходного текста и гаммы заменяются эквивалентными цифрами, которые затем складываются по модулю К, где К – число символов в алфавите, т.е.:

Ri = (Si + G) mod (К – 1),

где Ri, Si, G – символы зашифрованного текста, исходного текста и гаммы соответственно.

2. Двоичное гаммирование. Символы исходного текста и гаммы представляются в виде двоичного кода, затем соответствующие разряды складываются по модулю 2. Вместо сложения по модулю 2 при гаммиро- вании можно использовать другие логические функции, необходимым тре- бованием к которым является свойство обратимости преобразования.

Пример шифрования двоичным гаммированием представлен в таб-

лице:

Шифруемый текст Б У Д Ь …

 010010 100000 110010 100100

Знаки гаммы 7 1 8 2 …

 000111 000001 001000 000010

Шифрованный текст 010101 100001 111010 100110

Ход работы

Получите индивидуальное задание у преподавателя. Выберите метод получения гаммы шифра (псевдослучайной последовательности чисел). Реализуйте программный модуль в соответствии с полученным заданием. При реализации необходимо учесть следующие моменты:

1) предусмотреть возможность задания пользователем гаммы шифра;

2) предусмотреть визуализацию всех пользовательских настроек;

3) для удобства тестирования и взаимодействия с другими модулями реализовать файловый ввод исходных данных и файловый вывод результа- та криптографического преобразования.

После реализации программного модуля выполните статистический анализ текста до криптографического преобразования и после него.

Варианты заданий

1. Модуль для шифрования текста гаммированием по модулю.

2. Модуль для расшифровывания текста гаммированием по модулю.

3. Модуль для шифрования текста двоичным гаммированием.

4. Модуль для расшифровывания текста двоичным гаммированием.

Содержание отчёта

1. Цель работы.

2. Задание.

3. Анализ задания.

4. Алгоритм преобразования.

5. Программа на алгоритмическом языке.

6. Тестовые запуски и статистический анализ.

7. Выводы по работе.

Контрольные вопросы

1. Гаммирование: основные определения.

2. Алгоритм шифрования текста методом гаммирования

3. Двоичное гаммирование: основные особенности.

Практическое занятие № 4

ШИФРОВАНИЕ С ПОМОЩЬЮ АНАЛИТИЧЕСКИХ ПРЕОБРАЗОВАНИЙ

Цель работы

Исследование шифров, основанных на аналитических преобразова-

ниях.

Теоретическая часть

Достаточно надежное закрытие информации может быть обеспечено при использовании для шифрования некоторых аналитических преобразо- ваний. Например, умножение матрицы на вектор по правилу:

 N

C  A  B;

 aij b j .

j 1

Если матрицу А = (аij) использовать в качестве ключа, а вместо ком- понента вектора В = (bj) подставить символы текста, то компоненты векто- ра С = (сj) будут представлять собой символы зашифрованного текста.

Пример.

Возьмем в качестве ключа квадратную матрицу третьего порядка

⎛14 8 ⎞

⎜ ⎟

A  ⎜ 8 5 2 ⎟

⎜ ⎟

⎜ ⎟

⎝ ⎠

Заменим буквы алфавита цифрами, соответствующими их порядко- вому номеру в алфавите: А–0, Б–1, В–2 и т.д. Тогда отрывку текста ВА- ТАЛА будет соответствовать 2, 0, 19, 0, 12, 0. По принятому алгоритму шифрования необходимо выполнить следующие действия:

⎛14 8 ⎞ ⎛ ⎞ ⎛ 85 ⎞

 ⎜ ⎟ ⎜ ⎟ ⎜ ⎟

C  A B  ⎜8 5 2⎟⎜ 0 ⎟  ⎜ 54 ⎟;

⎜ ⎟

⎜ 2 1 ⎟

⎜ ⎟

⎜ 19 ⎟

⎝ ⎠

⎜ ⎟

⎜ 25 ⎟

⎝ ⎠

⎛14 8 3⎞ ⎛ 0

⎞ ⎛ 96 ⎞

⎜

 ⎜

⎟ ⎜ ⎟ ⎜ ⎟

⎟ ⎜ ⎟ ⎜ ⎟

C  A  B  ⎜8 5 2 ⎟ ⎜ 12 ⎟  ⎜ 60 ⎟.

⎜ ⎟ ⎜

⎜ 2 1 ⎟ ⎜ 0

⎟ ⎜ ⎟

⎟ ⎜ ⎟

⎠ ⎝ ⎠

При этом шифрованный текст будет иметь вид: 85, 54, 25, 96, 60, 24.

Дешифрование осуществляется с использованием указанного прави- ла умножения матрицы на вектор, только в качестве ключа берется матри- ца, обратная той, с помощью которой проводится зашифровывание, а в ка- честве вектора-сомножителя – соответствующие фрагменты символов за- крытого текста; тогда значениями вектора-результата будут цифровые эк- виваленты знаков открытого текста.

Матрицей, обратной данной А, называется матрица А-1, получаю- щаяся из присоединения матрицы делением всех ее элементов на опреде- литель данной матрицы. Присоединенной называется матрица, составлен- ная из алгебраических дополнений Аij, к элементам данной матрицы, кото- рые вычисляются по формуле

Aij

  1i j  ,

где

ij

– определитель матрицы, получаемой вычеркиванием i-й строки и

j- го столбца исходной матрицы.

Определителем матрицы называется алгебраическая сумма n! чле- нов (для определителя n-го порядка), составленная следующим образом: членами служат всевозможные произведения n элементов матрицы, взятых по одному в каждой строке и в каждом столбце; причем член суммы берет- ся со знаком «+», если его индексы составляют четную подстановку, и со знаком «–» – в противоположном случае. Для матрицы третьего порядка определитель вычисляется следующим образом:

 = a11a22a33 + a12a23а31 + а13а21а32 – а11а23а32 – а12а21а33 – а13а22а31.

Процесс раскрытия выглядит так:

⎛ 1  2

1 ⎞ ⎛85 ⎞

⎛1 85  2  54  1 25 ⎞ ⎛ 2 ⎞

A1  C ⎜ 2 5

 4⎟⎜54 ⎟

⎜ ⎟ ⎜ ⎟

2  85  5  54  4  25 

 ⎜

⎟ ⎜ ⎟  ⎜

⎟ ⎜ 0 ⎟;

⎜ 1  4

6 ⎟ ⎜ 25⎟

⎜1 85  4  54  6  25 ⎟

⎜19⎟

⎝ ⎠ ⎝ ⎠ ⎝ ⎠ ⎝ ⎠

⎛ 1  2

1 ⎞ ⎛96 ⎞ ⎛1 96  2  60  1 24

⎞ ⎛ 0 ⎞

1 C ⎜

⎟ ⎜ ⎟ ⎜

⎟ ⎜ ⎟

A   ⎜ 2 5  4⎟ ⎜ 60 ⎟  ⎜ 2  96  5  60  4  24⎟  ⎜12⎟.

⎜ 1  4

6 ⎟ ⎜ 24⎟

⎜1 96  4  60  6  24 ⎟

⎜ 0 ⎟

⎝ ⎠ ⎝ ⎠ ⎝ ⎠ ⎝ ⎠

Таким образом, получена последовательность знаков раскрытого тек- ста 2, 0, 19, 0, 12, 0, что соответствует исходному тексту.

Ход работы

Получите индивидуальное задание у преподавателя.

Реализуйте программный модуль в соответствии с полученным зада-

После реализации программного модуля выполните статистический

анализ текста до криптографического преобразования и после него.

Варианты заданий

1. Модуль для посимвольного шифрования текста.

2. Модуль для посимвольного расшифровывания текста.

3. Модуль для шифрования текста биграммами.

4. Модуль для расшифровывания текста биграммами.

Содержание отчёта

1. Цель работы.

2. Задание.

3. Анализ задания.

4. Алгоритм преобразования.

5. Программа на алгоритмическом языке.

6. Тестовые запуски и статистический анализ.

7. Выводы по работе.

Контрольные вопросы

1. В чем особенности метода аналитических преобразований.

2. Отличия метода посимвольного шифрования и шифрования текста биграммами.

3. Статистический анализ выполненного задания (минимум три при- мера).

4. Алгоритм выполнения поставленной задачи.

**Практическое занятие № 7 РАЗРАБОТКА И ИССЛЕДОВАНИЕ КРИПТОАЛГОРИТМА НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ СКРЕМБЛЕРА**

Цель работы

Познакомиться с простейшими методами потокового шифрования с использованием скремблеров.

Теоретическая часть

Суть скремблирования заключается в побитном изменении прохо- дящих через вычислительную систему потока данных. Практически един- ственной операцией, используемой в скремблерах, является XOR – "по- битное исключающее ИЛИ". Параллельно прохождению информационно- го потока в скремблере по определенному правилу генерируется поток бит – кодирующий поток. Как прямое, так и обратное шифрование осуществля- ется наложением кодирующей последовательности на исходную с исполь- зованием операции XOR.

Генерация кодирующей последовательности бит производится цик- лически из небольшого начального объема информации – ключа – по сле- дующему алгоритму. Из текущего набора бит выбираются значения опре- деленных разрядов и складываются с помощью операции XOR между со- бой. Все разряды сдвигаются на 1 бит, а только что полученное значение ("0" или "1") помещается в освободившийся самый младший разряд. Зна- чение, находившееся в самом старшем разряде до сдвига, добавляется в кодирующую последовательность, становясь очередным ее битом (рису- нок).

 Ход работы

Получите индивидуальное задание у преподавателя. Реализуйте про- граммный модуль в соответствии с полученным заданием. При реализации необходимо учесть следующие моменты:

1) кодирование строки осуществляется последовательным примене- нием скремблера к каждому биту каждого ее символа;

2) определить период повторения последовательности, генерируемой заданным скремблером, для нескольких начальных значений ключа, вы- бранных случайным образом в диапазоне (0 – 255).

Варианты заданий

1. x8  x7  x6  x3  x2  1

2. x 9

 x 3  1

3. x10  x5  x4  x2  1

4. x5  x 4  x 2  1

5. x11  x5  x 2  1

6. x12  x7  x3  x  1

7. x8  x 6  x 2  1

Содержание отчёта

1. Цель работы.

2. Задание.

3. Анализ задания.

4. Алгоритм преобразования.

5. Программа на алгоритмическом языке.

6. Тестовые запуски и статистический анализ.

7. Выводы по работе.

Контрольные вопросы

1. Назовите преимущества и недостатки использования скремблера.

2. Укажите свойства, которыми должна обладать псевдослучайная последовательность, генерируемая скремблером.

3. Для каких целей используют скремблеры и дескремблеры?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 8 АЛГОРИТМ БЛОЧНОГО ШИФРОВАНИЯ ДАННЫХ ГОСТ 28147-89**

Цель работы: научиться реализовывать на выбранном языке программирования алгоритм блочного шифрования данных ГОСТ 28147-89.

ОСНОВНЫЕ ПОНЯТИЯ

Криптоалгоритм ГОСТ 28147-89 является стандартом блочного шифрования РФ. Алгоритм имеет следующие параметры:

 размер блока – 64 бита;

 размер ключа – 256 бит. Ключ представляется как массив из восьми 32-битных подключей K={K0,K1,…,K7};

 количество раундов – 32 в режиме шифрования и 16 в режиме выработки имитовставки;

 количество S-блоков – 8. Каждый S-блок (в терминах стандарта

– узел замены) содержит 16 четырехбитных значений, представляющих собой произвольную перестановку чисел от 0 до 15. Совокупность всех S-блоков можно представить в виде матрицы размером (816), называемой в терминах стандарта таблицей замен.

Алгоритм является вариантом сети Фейстеля, что предполагает одинаковую структуру каждого раунда (в терминах алгоритма – основного шага). Входом очередного раунда является выход предыдущего. Функция раунда, которая применяется к правой половине входного блока и всех промежуточных результатов состоит из следующих этапов:

 сложение правой половины блока с подключом раунда по модулю 232. При зашифровании подключи из массива К используются в следующем порядке: K0,K1,…,K7, K0,…,K7,K0,…,K7,K7,K6,…,K0, при расшифровании – в обратном порядке;

 подстановка через S-блок. 32-битовый результат сложения с подключом раунда представляется как 8 четырехбитовых фрагментов, каждый из которых заменяется тем элементом соответствующего ему S-блока, номер которого равен значению данного фрагмента;

 циклический сдвиг результата подстановки на 11 разрядов влево.

Результат вышеизложенных преобразований складывается по модулю 2 с левой половиной выхода предыдущего раунда. Сумма становится правой половиной выхода текущего раунда. Левой половиной становится правая половина выхода предыдущего раунда.

Для зашифрования 64-битового блока открытого текста выполняются 32 раунда. При этом в соответствии со структурой сети Фейстеля, перестановка левой и правой половин блока после последнего раунда не производится. Расшифрование производится по той же схеме, что и зашифрование, за исключением того, что подключи используются в обратном порядке.

Стандарт определяет три режима шифрования и один режим выработки имитовставки. К режимам шифрования относятся:

 простая замена. Данный режим предполагает независимое зашифрование/расшифрование 64-битных блоков открытого текста/шифртекста с помощью рассмотренного выше алгоритма. Поскольку алгоритм обрабатывает только 64-битные блоки, то в случае зашифрования неполного последнего блока необходимо организовать его дополнение. Сам стандарт не задает способ дополнения. Существуют несколько различных подходов к выполнению данной операции. Например, в качестве дополнения может использоваться случайная последовательность нулей и единиц. Последний байт дополнения должен содержать его размер, т.е. количество байт, которые необходимо отбросить при расшифровании. В том случае, если изначально размер открытого текста кратен 8 байтам, производится дополнение целым блоком. Еще одним вариантом является способ, предложенный в стандарте PKCS#7, разработанном компанией RSA Security, Inc. Согласно ему все дополняемые байты (вплоть до полного блока) заполняются одним и тем же числом, представляющим собой размер этого дополнения в байтах. Естественно, они также должны быть отброшены при расшифровании;

 гаммирование. Этот режим позволяет шифровать данные размером менее 64 бит. Базовый алгоритм используется в составе рекуррентного генератора псевдослучайных чисел. Вырабатываемая им гамма накладывается на блок открытого текста (или шифртекста при расшифровании) с помощью операции поразрядного

«исключающего или» так же, как и в потоковых шифрах. Генератор инициализируется с помощью произвольного начального 64-битового значения, называемого вектором инициализации или синхропосылкой. В начале синхропосылка зашифровывается с помощью базового алгоритма. Далее результат этого зашифрования разбивается на

старшую и младшую 32-битные части. Младшая часть складывается с константой 101010116 по модулю 232, а старшая с константой 101010316 по модулю (232 – 1) с дальнейшим прибавлением единицы. Результаты сложения вновь поступают на вход генератора и процесс повторяется. Одновременно эти две 32-битные части объединяются в 64-битный блок, который зашифровывается с помощью базового алгоритма. Полученное таким образом значение используется в качестве гаммы;

 гаммирование с обратной связью. Данный режим, так же как и предыдущий, предполагает наложение гаммы на блок открытого текста или шифртекста с помощью операции «исключающее или». Однако вырабатываемая здесь гамма зависит не только от ключа, но и от всего открытого текста. Режим также предполагает наличие синхропосылки. В качестве первого элемента гаммы используется зашифрованная базовым алгоритмом синхропосылка. Он складывается по модулю два с первым блоком открытого текста, в результате чего получается первый блок шифртекста. Для зашифрования второго блока открытого текста используется элемент гаммы, полученный в результате применения базового алгоритма к первому блоку шифртекста и т.д. Таким образом, на шифрование текущего блока влияют результаты шифрования всех предыдущих. Расшифрование производится аналогичным образом. В данном режиме, как и в предыдущем, базовый алгоритм всегда работает в режиме зашифрования.

СОДЕРЖАНИЕ РАБОТЫ

Разработать на выбранном языке программирования консольное или оконное приложение, реализующее алгоритм ГОСТ 28147-89 в следующих режимах шифрования:

 режим простой замены (с дополнением блоков);

 режим гаммирования;

 режим гаммирования с обратной связью.

Программа должна запрашивать имя входного и выходного файлов, ключ, вектор инициализации (синхропосылку), режим работы (зашифрование или расшифрование), режим шифрования.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое сеть Фейстеля? Каковы основные принципы работы блочных шифров, устроенных по принципу сети Фейстеля?

2. Назовите все режимы шифрования, определенные в ГОСТ 28147-89.

3. Каковы разрядности блока и ключа в алгоритме ГОСТ 28147- 89?

4. Что представляют собой таблицы замен (S-блоки) в алгоритме ГОСТ 28147-89?

5. Что представляет собой один раунд (основной шаг) алгоритма ГОСТ 28147-89?

6. Как может производиться дополнение неполных блоков в режиме простой замены?

7. Каковы недостатки режима простой замены?

8. Что собой представляет режим гаммирования?

9. Что собой представляет режим гаммирования с обратной связью?

10. Как функционирует схема зашифрования алгоритма ГОСТ 28147-89?

11. Как функционирует схема расшифрования алгоритма ГОСТ 28147-89?

12. Что такое синхропосылка?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 9 СИММЕТРИЧНОЕ ШИФРОВАНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ КРИПТОГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ MICROSOFT CRYPTOAPI И CRYPTOGRAPHY API: NEXT GENERATION**

Цель работы: ознакомиться с понятием криптопровайдера в интерфейсе CryptoAPI и провайдера алгоритма в интерфейсе Cryptography API: Next Generation, способами подключения к ним, получить навыки выполнения базовых криптографических преобразований: хэширования и симметричного шифрования.

ОСНОВНЫЕ ПОНЯТИЯ

Интерфейс CryptoAPI

В операционных системах (ОС) компании Microsoft, начиная с Windows 95, обеспечивается реализация шифрования, генерации ключей, создания и проверки цифровых подписей и других криптографических преобразований. Эти функции необходимы для работы операционной системы, однако ими может воспользоваться и любая прикладная программа. Для этого используется интерфейс прикладного программирования CryptoAPI.

Все современные операционные системы Windows поддерживают криптографический интерфейс CryptoAPI 2.0. Он содержит функции, осуществляющие базовые криптографические преобразования, а также дополнительные средства, такие как функции для работы с сертификатами X.509. Набор функций для осуществления базовых криптографических преобразований также называют CryptoAPI 1.0. Ниже будут рассмотрены некоторые функции именно этого интерфейса.

Все функции интерфейса CryptoAPI 1.0 содержатся в библиотеке advapi32.dll. Однако они выполняют лишь ряд вспомогательных операций и вызывают библиотеку, в которой непосредственно реализованы соответствующие криптографические преобразования. Такие библиотеки называются криптопровайдерами (Cryptographic Service Providers, CSP). Криптопровайдеры имеют стандартный набор из 23 обязательных и 2 необязательных функций. Программная часть криптопровайдера представляет собой dll-файл, подписанный

Microsoft; периодически Windows проверяет цифровую подпись, что исключает возможность подмены криптопровайдера. В составе Windows могут быть установлены криптопровайдеры разработанные не только Microsoft, но и сторонними производителями. Сведения обо всех установленных криптопровайдерах содержатся в системном реестре (HKLM\SOFTWARE\Microsoft\Cryptography\Defaults\Provider).

Каждый криптопровайдер поддерживает свою ключевую базу, которая представляется набором ключевых контейнеров. Каждый контейнер имеет имя, которое присваивается ему при создании, а затем используется для работы с ним. В ключевом контейнере сохраняется долговременная ключевая информация, например пары закрытый/открытый ключ для создания и проверки электронной подписи (ЭП). Сеансовые ключи для блочного или потокового шифрования в контейнере не хранятся.

Каждый криптопровайдер характеризуется собственным именем и типом. Имя это строка символов, по которой система распознает криптопровайдера. Например, базовый криптопровайдер, имеющийся в составе ОС Windows любой версии называется «Microsoft Base Cryptographic Provider v1.0». Тип криптопровайдера это целое число (тип DWORD), значение которого идентифицирует набор поддерживаемых алгоритмов цифровой подписи и обмена сеансовыми ключами. Криптопровайдер «Microsoft Base Cryptographic Provider v1.0» имеет тип 1, что соответствует символической константе PROV\_RSA\_FULL. В криптопровайдерах этого типа используется алгоритм RSA и для создания электронной подписи и для обмена сеансовыми ключами. В системе могут быть установлены криптопровайдеры разных типов. Также может быть установлено несколько криптопровайдеров одного типа.

В системном реестре содержится информация не только об именах криптопровайдеров и их типах, но и о том какой криптопровайдер использовать, если пользователь при вызове соответствующей функции не определил конкретное имя, а указал только требуемый ему тип. Такие криптопровайдеры называются используемыми по умолчанию. Например, для типа 1, в ОС, предшествующих Windows 2000, криптопровайдером по умолчанию являлся «Microsoft Base Cryptographic Provider v1.0». Начиная с Windows 2000, в состав ОС включены еще 2 криптопровайдера того же типа: «Microsoft Enhanced Cryptographic Provider» и «Microsoft Strong Cryptographic Provider», один из которых назначается криптопровайдером типа 1 по умолчанию (обычно «Microsoft Strong Cryptographic Provider»).

Криптопровайдеры Microsoft 1-го типа поддерживают уже устаревшие на данный момент алгоритмы шифрования RC2, RC4, DES, хэширования MD5, SHA-1. Современные алгоритмы: шифрования AES (с длинами ключа 128, 192, 256 бит) и хэширования SHA-2 (с длинами хэш-кода 256, 384, 512 бит) поддерживаются только криптопровайдерами типа 24 (PROV\_RSA\_AES). Начиная с версии Windows XP SP3, в состав операционных систем Microsoft включается единственный криптопровайдер этого типа «Microsoft Enhanced RSA and AES Cryptographic Provider». Он, также как и криптопровайдеры типа 1, использует алгоритм RSA для обмена ключами и создания ЭП. Расположены при этом все вышеперечисленные криптопровайдеры в составе одного файла – rsaenh.dll.

Для использования функции CryptoAPI в прикладной программе, разработчик должен объявить ее в программе как внешнюю, с указанием файла, в котором она находится. Чтобы облегчить процесс разработки, объявления необходимых типов, символических констант (таких как PROV\_RSA\_AES) и прототипов функций объединили в заголовочном файле wincrypt.h, который вошел в состав пакета Platform (Windows) SDK. По умолчанию пакет устанавливается в составе таких сред программирования, как С++ Builder и Visual Studio. Также он доступен для бесплатной загрузки на сайте компании Microsoft. Для использования в программе функций CryptoAPI необходимо включить в ее текст директивой #include содержимое файла wincrypt.h.

Рассмотрим функции CryptoAPI, необходимые для выполнения задания. Все описания и примеры будут приведены для языка программирования C/C++, применительно к консольному приложению (в оконных приложениях отличия только в способе вывода диагностических сообщений). Применяемые типы данных определены в файлах wincrypt.h и windef.h. Последний традиционно включается в текст в числе многих файлов, необходимых для разработки приложения Win32 API, при указании директивы #include <windows.h>.

В рассматриваемых далее примерах будут использоваться переменные, описание которых приведено ниже:

HCRYPTPROV hProv; //дескриптор криптопровайдера HCRYPTKEY hKeyH; //дескриптор ключа, созданного из хэш-кода HCRYPTHASH hHash; //дескриптор хэш-объекта

BYTE aBuf[512]; //буфер для данных (размер произвольный, кратный 16)

DWORD dwBuflen; //длина буфера

TCHAR szPass[100]; //строка для пароля

Назначение переменных будет пояснено по мере рассмотрения тех функций CryptoAPI, которые понадобятся при выполнении данной лабораторной работы.

Полное описание этих и других функций можно найти в разделе MSDN, посвященном использованию CryptoAPI (https://msdn.microsoft.com/en-us/library/windows/desktop/aa380256%28 v=vs.85%29.aspx).

CryptAcquireContext

Эта функция выполняет подключение к криптопровайдеру. Ее прототип имеет вид:

BOOL WINAPI CryptAcquireContext(HCRYPTPROV \*phProv,

LPCTSTR pszContainer, LPCTSTR pszProvider, DWORD dwProvType, DWORD dwFlags);

В Win32 API (в том числе и в CryptoAPI) традиционно библиотечные функции, имеющие в качестве параметров строки, представлены в двух вариантах: с однобайтными ANSI-строками и

«широкими» строками UNICODE. Здесь показан обобщенный прототип функции (макрос), который после обработки программы препроцессором будет заменен, в соответствии с настройками проекта, на нужный вариант (с буквой «A» или «W» в конце).

Функция возвращает дескриптор криптопровайдера в параметре phProv. В дальнейшем этот параметр будет передаваться другим функциям.

Параметр pszContainer содержит имя ключевого контейнера. Если приложение не будет использовать ключевой контейнер, то параметр задается как NULL. В частности, этот параметр должен быть нулевым, если приложение не будет производить экспорт/импорт сеансовых ключей. В противном случае имя контейнера необходимо задавать.

Параметр pszProvider содержит имя того криптопровайдера, к которому производится подключение. Параметр dwProvType определяет тип криптопровайдера. При вызове функции вместо него можно указывать символические константы, например, PROV\_RSA\_AES. Если в качестве параметра pszProvider указать NULL, то будет осуществлено подключение к криптопровайдеру, используемому по умолчанию для типа, указанного в параметре dwProvType.

Параметр dwFlags содержит значения флага (здесь и при описании некоторых других функций будут приведены только некоторые

значения флагов; полное описание функций можно найти в соответствующей литературе или библиотеке MSDN). Если задать его равным константе CRYPT\_VERIFYCONTEXT, то получение контекста произойдет без подключения к контейнеру ключей. Для создания нового контейнера ключей с именем, заданным в параметре pszContainer используется флаг CRYPT\_NEWKEYSET. Нулевое значение флага используется для подключению к уже существующему контейнеру.

Все функции CryptoAPI имеют логический тип (BOOL) и возвращают TRUE в случае успешного завершения и FALSE в противном случае. Поэтому при каждом вызове необходимо проверять возвращаемое функцией значение. В случае ошибки ее код можно узнать с помощью функции GetLastError. Один из возможных вариантов обработки ошибки выполнения функции, получающей контекст без подключений к ключевому контейнеру, приведен ниже.

if (!CryptAcquireContext(&hProv, NULL, NULL, PROV\_RSA\_AES, CRYPT\_VERIFYCONTEXT))

{

\_tprintf(TEXT("Ошибка подключения к\ криптопровайдеру. Код ошибки:0x%X"), GetLastError());

}

Данный вариант вызова функции CryptAcquireContext выполнит подключение к криптопровайдеру типа 24 по умолчанию (то есть

«Microsoft Enhanced RSA and AES Cryptographic Provider»).

Другой вариант вызова функции производит подключение к контейнеру ключей «my\_cont», а в случае отсутствия создает его заново:

TCHAR szBuf[100];

if (!CryptAcquireContext(&hProv,TEXT("my\_cont"),

NULL, PROV\_RSA\_AES, 0))

{

DWORD err=GetLastError(); if(err!=NTE\_BAD\_KEYSET)// Прочая ошибка

{

\_tprintf(TEXT("Ошибка подключения к\ криптопровайдеру. Код ошибки:0x%X"), err);

}

else // Создаем контейнер ключей «my\_cont»

if (!CryptAcquireContext(&hProv, TEXT("my\_cont"),

NULL, PROV\_RSA\_AES, CRYPT\_NEWKEYSET))

{

\_tprintf(TEXT("Ошибка создания контейнера ключей.\ Код ошибки:0x%X"), GetLastError());

}

}

После того, как произведено подключение к нужному криптопровайдеру, для выполнения задачи симметричного шифрования необходимо получить сессионный ключ. Он может быть импортирован или сгенерирован. В свою очередь генерация ключа может быть произведена двумя способами: с помощью генератора псевдослучайных чисел или на основе хэшированной парольной фразы. Рассмотрим более подробно второй способ.

Однонаправленные хэш-функции выполняют свертку входных данных произвольной длины в битовую последовательность фиксированной длины, называемую хэш-кодом или дайджестом. При этом определение сообщения (прообраза), которое приводит к созданию данного хэш-кода, является трудновыполнимой (с точки зрения необходимых вычислительных ресурсов) задачей. А изменение всего лишь одного бита исходного сообщения приводит к кардинальным изменениям в его хэш-коде. Таким образом, пользователь, придумав парольную фразу и получив ее дайджест, использует его в качестве ключа симметричного шифрования.

В криптопровайдре «Microsoft Enhanced RSA and AES Cryptographic Provider» реализована поддержка алгоритма хэширования SHA-2 с переменной длиной дайджеста: 256, 384, или

512 бит. Для создания ключа с помощью хэш-кода используется функция CryptDeriveKey. Однако перед ее вызовом необходимо, чтобы в памяти был создан хэш-объект, содержащий какой-либо хэш-код. Поэтому вначале рассмотрим функции для работы с хэш-объектом.

CryptCreateHash

Эта функция создает в памяти хэш-объект и имеет следующий прототип:

BOOL WINAPI CryptCreateHash(HCRYPTPROV hProv,

ALG\_ID Algid, HCRYPTKEY hKey,

DWORD dwFlags, HCRYPTHASH \*phHash);

Через параметр hProv в функцию передается дескриптор криптопровайдера. Параметр Algid представляет собой идентификатор используемого алгоритма хэширования. При использовании криптопровайдера типа 24 целесообразно использовать один из следующих алгоритмов: SHA-256, SHA-384, SHA-512. Соответственно определены следующие константы: CALG\_SHA\_256, CALG\_SHA\_384, CALG\_SHA\_512. Именно их можно указывать при вызове функции в качестве второго параметра. Однако их описание может отсутствовать в файле wincrypt.h, если он входит в состав Platform SDK устаревшей версии. В этом случае константы можно описать вручную:

#define CALG\_SHA\_256 0x800C #define CALG\_SHA\_384 0x800D #define CALG\_SHA\_512 0x800E

Параметр hKey содержит дескриптор ключа и используется в том случае, если должен быть создан код аутентичности сообщения (HMAC). В нашем случае этот параметр должен быть равен нулю. Параметр dwFlags содержит флаги и должен быть равен нулю. Функция копирует дескриптор созданного хэш-объекта по адресу, переданному через параметр phHash.

Приведем пример вызова функции. В этом примере и во всех последующих для краткости будут приведены только сами вызовы функций без проверки успешности их завершения. Однако в реальном приложении эти проверки необходимо осуществлять всегда, аналогично варианту, рассмотренному для функции CryptAcquireContext.

CryptCreateHash(hProv, CALG\_SHA\_256, 0, 0, &hHash);

Функция CryptCreateHash создает пустой хэш-объект. Подать данные на вход хэш-объекта и собственно вычислить хэш-код позволяет функция CryptHashData.

CryptHashData

Функция позволяет добавлять данные к объекту хэш-функции. Она может вызываться несколько раз, если данные, от которых вычисляется хэш, разбиты на блоки. Прототип функции имеет вид:

BOOL WINAPI CryptHashData(HCRYPTHASH hHash,

BYTE \*pbData, DWORD dwDataLen, DWORD dwFlags);

В качестве параметра hHash передается дескриптор хэш-объекта, созданный функцией CryptCreateHash. Параметр pbData содержит указатель на хэшируемые данные, а dwDataLen содержит размер этих данных в байтах. Параметр dwFlags должен быть равен нулю.

Приведем пример вызова функции, осуществляющего хэширование некоторого пароля, хранящегося в строке szPass (описание см. выше).

...................................................

//Получение каким-либо образом пароля и занесение его в szPass.

...................................................

CryptHashData(hHash, (PBYTE)szPass,

\_tcslen(szPass)\*sizeof(TCHAR), 0);

После того, как произведено хэширование парольной фразы, необходимо вызвать функцию CryptDeriveKey (описание см. ниже). После формирования ключа хэш-объект должен быть уничтожен функцией CryptDestroyHash.

CryptDestroyHash

Функция уничтожает хэш-объект, ранее созданный функцией

CryptCreateHash. Ее прототип имеет вид:

BOOL WINAPI CryptDestroyHash(HCRYPTHASH hHash);

Единственный параметр представляет собой дескриптор уничтожаемого объекта. Пример вызова:

CryptDestroyHash(hHash);

CryptDeriveKey

Функция создает сеансовый ключ на базе хэшированного пароля.

Прототип функции имеет вид:

BOOL WINAPI CryptDeriveKey(HCRYPTPROV hProv,

ALG\_ID Algid, HCRYPTHASH hBaseData, DWORD dwFlags, HCRYPTKEY \*phKey);

Параметр hProv содержит дескриптор криптопровайдера, полученный с помощью функции CryptAcquireContext. Параметр Algid содержит идентификатор того алгоритма шифрования, для которого создается ключ. При использовании криптопровайдера типа 24 целесообразно использовать следующие варианты: AES-128, AES-192, AES-256. Соответственно определены константы: CALG\_AES\_128, CALG\_AES\_192, CALG\_AES\_256.

Параметр hBaseData содержит дескриптор хэш-объекта, содержащего хэшированный пароль. Причем после создания сеансового ключа в хэш-объект запрещено добавлять новые данные Параметр dwFlags, содержащий значения флагов, можно установить равным нулю. Функция копирует дескриптор созданного ключа по адресу, переданному через параметр phKey. Пример вызова:

CryptDeriveKey(hProv, CALG\_AES\_128, hHash, 0, &hKeyH);

Данный вызов создаст сессионный ключ, представляющий собой объект в защищенной области памяти, доступ к которому осуществляется через дескриптор. При этом объект содержит в себе не только сам ключ, но и параметры шифрования: алгоритм, режим (по умолчанию CBC), вектор инициализации или синхропосылку (по умолчанию нулевой), размер блока и т.д. Непосредственно в открытом виде ключ приложению недоступен.

После того, как создан сеансовый ключ, можно осуществлять зашифрование или расшифрование с помощью функций CryptEncrypt или CryptDecrypt (описание см. ниже), передавая им в качестве одного из параметров полученный дескриптор. После шифрования ключ необходимо уничтожить функцией CryptDestroyKey. В дальнейшем такой ключ может быть воссоздан, если предоставить для хэширования первоначальную парольную фразу.

CryptDestroyKey

Функция освобождает ранее определенный дескриптор ключа, созданного любым из двух способов или дескриптор ключевой пары. Ее прототип имеет вид:

BOOL WINAPI CryptDestroyKey(HCRYPTKEY hKey);

Если параметром функции является дескриптор сессионного ключа, то при вызове уничтожается сам ключ. Если дескриптор ключевой пары, то происходит только его освобождение, а сама пара остается в контейнере.

Пример вызова функции:

CryptDestroyKey(hKeyH);

CryptEncrypt

Функция осуществляет зашифрование данных. Прототип имеет вид:

BOOL WINAPI CryptEncrypt(HCRYPTKEY hKey,

HCRYPTHASH hHash,

BOOL Final, DWORD dwFlags, BYTE \*pbData,

DWORD \*pdwDataLen, DWORD dwBufLen);

В параметре hKey передается дескриптор ключа, необходимый для шифрования. Этот ключ также определяет алгоритм шифрования. Параметр hHash используется, если исходные данные одновременно шифруются и хэшируются. В нашем случае этот параметр следует установить равным нулю. 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. В противном случае передается значение FALSE. Параметр dwFlags не используется, и на его месте следует указать ноль. Параметр pbData – указатель на буфер, в котором содержатся данные для зашифрования. Зашифрованные данные по завершении работы функции помещаются в тот же буфер. Через параметр pdwDataLen в функцию при вызове передается размер исходных данных, а по завершении возвращается размер зашифрованных. dwBufLen – размер выходного буфера, для блочных шифров может быть больше, чем pdwDataLen.

Если используется блочный шифр AES, то максимальное значение параметра pdwDataLen при вызове функции составляет (dwBufLen - 16). Это связано с тем, что шифрование по умолчанию ведется в режиме CBC. Размер блока в данной реализации алгоритма всегда равен 128 битам. При зашифровании последний блок (Final устанавливается в TRUE) всегда конкатенируется с дополнением, размер которого в данном случае может составлять от 1 до 16 байт. Реальный размер зашифрованного блока, используемый для его записи в выходной файл, возвращается также через параметр pdwDataLen. Если размер исходных данных при вызове превышает 128 бит, то функция самостоятельно делит их на блоки требуемого алгоритмом размера. При этом размер любого входного блока, кроме последнего, должен быть кратен 16 байтам.

Пример вызова функции для зашифрования не последнего блока:

dwBuflen = 496;

CryptEncrypt(hKeyH, 0, FALSE,0, aBuf, &dwBuflen, 512);

CryptDecrypt

Функция осуществляет расшифрование данных. Прототип имеет вид:

BOOL WINAPI CryptDecrypt(HCRYPTKEY hKey,

HCRYPTHASH hHash,

BOOL Final, DWORD dwFlags, BYTE \*pbData,

DWORD \*pdwDataLen);

Все параметры функции имеют тоже значение, что и для функции CryptEncrypt. Шифртекст передается через параметр pbData, а его длина через pdwDataLen. Функция возвращает реальную длину расшифрованного блока также через параметр pdwDataLen. Размер выходных данных, в отличие от шифрующей функции, не указывается. Это связано с тем, что размер расшифрованных данных может только уменьшаться и отведенного при вызове размера буфера будет вполне достаточно.

Пример вызова функции для расшифрования не последнего блока. Переменная dwBuflen должна содержать размер передаваемых через буфер aBuf данных:

CryptDecrypt(hKeyH, 0, FALSE, 0, aBuf, &dwBuflen);

CryptReleaseContext

После завершения работы с функциями CryptoAPI, необходимо освободить контекст криптопровайдера, полученный ранее с помощью функции CryptAcquireContext. Для этого вызывается функция CryptReleaseContext. Ее прототип имеет вид:

BOOL WINAPI CryptReleaseContext(HCRYPTPROV hProv,

DWORD dwFlags);

Параметр hProv – дескриптор освобождаемого контекста криптопровайдера. Параметр dwFlags должен равняться нулю. Пример вызова функции:

CryptReleaseContext(hProv, 0);

Интерфейс Cryptography API: Next Generation

Криптографический интерфейс CryptoAPI, частично описанный выше, поддерживался и продолжает поддерживаться всеми версиями операционных систем семейства Windows. Однако, начиная с версии Windows Vista среди настольных ОС и Windows 2008 Server среди

серверных, появилась поддержка так называемого криптографического интерфейса следующего поколения: Cryptography API: Next Generation (сокращенно CNG).

Сравнивая интерфейс CNG с предшественником можно сказать, что по многим параметрам они схожи, но есть и различия. В CryptoAPI 1.0 центральным понятием является криптопровайдер (CSP). Существует относительно большое число типов криптопровайдеров, распространяемых для ОС Windows. Пользователю необходимо получить контекст одного из криптопровайдеров нужного типа и использовать возвращенный ему дескриптор для получения доступа к его функциям.

В CNG также имеется понятие провайдера. Однако порядок действий с ними несколько иной. Пользователь непосредственно может получать дескриптор нужного ему криптоалгоритма, не задумываясь к какому типу относится предоставляющий его провайдер. По сути, в CNG осталось всего четыре типа провайдеров. В стандартную поставку Windows входят обычно четыре CNG-провайдера, каждый из которых относится к соответствующему типу:

 «Microsoft Primitive Provider»;

 «Microsoft Smart Card Key Storage Provider»;

 «Microsoft Software Key Storage Provider»;

 «Microsoft SSL Protocol Provider».

Программы, которые используют средства первого и последнего из перечисленных провайдеров могут работать как в режиме ядра, так и пользовательском режиме. Использование остальных провайдеров возможно только в пользовательском режиме. Сведения об имеющихся провайдерах CNG можно получить в ветви реестра HKLM\SYSTEM\CurrentControlSet\Control\Cryptography. В частности, там содержится информация о том, какой из провайдеров CNG поддерживает выполнение того или иного криптоалгоритма, а также информация о месторасположении провайдеров.

Из названий вышеперечисленных провайдеров становится ясным их назначение. «Microsoft Primitive Provider» поддерживает выполнение функций так называемых криптографических примитивов: генерации ключей и случайных чисел, шифрования, хэширования, экспорта и импорта ключей. Провайдер «Microsoft Software Key Storage Provider» отвечает за хранение ключей асимметричного шифрования и цифровой подписи. Если такие ключи должны храниться на смарт-картах, то используется «Microsoft Smart Card Key Storage Provider». Ну и, наконец, провайдер «Microsoft SSL

Protocol Provider» применяется в приложениях, реализующих протоколы безопасной передачи данных SSL и TLS.

Далее в данной работе будут рассматриваться особенности применения криптографических примитивов для симметричного шифрования данных. Все содержимое «Microsoft Primitive Provider» для работы в пользовательском режиме находится в файле bcryptprimitives.dll. По сути это набор провайдеров отдельных алгоритмов. «Microsoft Primitive Provider» поддерживает работу всех тех же алгоритмов, которые реализованы в криптопровайдерах CryptoAPI от Microsoft. Кроме того, добавлена поддержка реализации алгоритмов Диффи-Хеллмана и DSA с использованием эллиптических кривых (ECDH, ECDSA), а также новых алгоритмов генерации псевдослучайных чисел.

Для использования криптографических примитивов пользователь напрямую работает не с указанной выше библиотекой, а с файлом так называемого маршрутизатора примитивов (primitive router). Этот файл содержит функции, которые получают доступ к одному из криптографических интерфейсов: асимметричного, симметричного шифрования, хэширования и других, реализованных в библиотеке bcryptprimitives.dll. В качестве маршрутизатора при работе с примитивами в пользовательском режиме используется библиотека bcrypt.dll.

Соответственно, все функции, реализованные в данном файле, имеют префикс BCrypt (буква «B» означает «базовый). Например, BCryptOpenAlgorithmProvider или BCryptEncrypt. Для доступа к сервисам других провайдеров в основном используются функции, реализованные в файле ncrypt.dll, и имеющие в своем названии префикс NCrypt (буква «N» означает «новый»). Также в данном файле реализованы функции для работы с «Microsoft SSL Protocol Provider» и имеющие, соответственно, префикс Ssl.

Рассмотрим более подробно, как выполняется симметричное шифрование данных с использованием криптографических примитивов CNG. Для этого необходимо выполнить следующие действия:

1. Загрузить и инициализировать провайдер нужного алгоритма шифрования.

2. Если необходимо, установить требуемые параметры алгоритма.

3. Определить размер ключевого объекта и выделить для него память.

4. Сгенерировать (или импортировать) сеансовый ключ.

5. Если необходимо, установить требуемые параметры ключа.

6. Зашифровать/расшифровать данные.

7. Экспортировать (если необходимо) и уничтожить ключ.

8. Закрыть провайдер алгоритма шифрования.

Генерацию ключа, также как и в CryptoAPI, можно выполнять с помощью генератора случайных чисел или на основе хэшированной парольной фразы. В данной лабораторной работе не рассматриваются вопросы экспорта и импорта ключей. Поэтому для генерации сеансового ключа будет применяться второй способ, из чего следует, что перед выполнением пункта 1, дополнительно должны быть выполнены следующие действия:

1. Загрузить и инициализировать провайдер нужного алгоритма хэширования.

2. Определить размер будущего хэш-объекта и выделить для него память.

3. Создать хэш-объект.

4. Поместить в хэш-объект парольную фразу.

5. Извлечь вычисленный хэш-код, который в дальнейшем будет передан функции, генерирующей сеансовый ключ.

6. Уничтожить хэш-объект.

7. Закрыть провайдер алгоритма хэширования.

Далее будут рассмотрены функции, которые потребуются для реализации приведенных выше алгоритмов. Все прототипы функций будут приведены на языке C/C++, как они и указаны в документации. В проектах Win32 для получения доступа к функциям работы с примитивами необходимо загрузить библиотеку bcrypt.dll. Для этого можно использовать библиотеку импорта bcrypt.lib, указав этот файл в свойствах проекта (Проект Свойства  Свойства конфигурации  Компоновщик  Ввод  Дополнительные зависимости) или указав строку в тексте программы:

#pragma comment(lib, " bcrypt.lib")

В файле исходного кода необходимо включить в текст директивой #include содержимое файлов windows.h и bcrypt.h. Причем включать файлы нужно именно в этом порядке, так как bcrypt.h, содержащий прототипы функций, константы, макросы, типы данных модуля BCrypt, использует в свою очередь некоторые типы данных, определенные в файлах, включение которых производится с помощью windows.h.

Все функции CNG, оперирующие строковыми параметрами, существуют только в одном варианте: с Unicode-строками. Поэтому строковые переменные нужно определять типом WCHAR (или TCHAR

и указывать в свойствах проекта, что используется Unicode), а перед константами ставить префикс “L” (или использовать макрос TEXT). Однако необходимость непосредственного оперирования ими возникает достаточно редко, так как для всех используемых функциями строковых констант в bcrypt.h определены символические константы.

Что касается специальных типов данных, определенных в bcrypt.h, то для описания приведенных ниже функций мы будем использовать переменные всего трех из них:

BCRYPT\_ALG\_HANDLE hAlg; BCRYPT\_KEY\_HANDLE hKey; BCRYPT\_HASH\_HANDLE hHash;

Переменная hAlg будет использоваться в качестве дескриптора провайдера алгоритма, hKey в качестве дескриптора ключа, а hHash в качестве дескриптора хэш-объекта. Причем все три типа определены в файле bcrypt.h одинаково, как указатели на тип VOID. Что касается остальных типов параметров описываемых функций, то они являются стандартными типами Win32 API: DWORD, PBYTE.

Все функции CNG возвращают 32-битное значение типа NTSTATUS, которое представляет собой код ошибки ядра. Возможные значения определены в файле ntstatus.h. По структуре этот код схож со значением, возвращаемым функцией GetLastError. Одним из вариантов определения успешности выполнения функции, является использование макроса BCRYPT\_SUCCESS, определенного в файле bcrypt.h:

#define BCRYPT\_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

Если значение, возвращаемое макросом BCRYPT\_SUCCESS истинно, то функция считается выполненной успешно. Проверка на неотрицательность в макросе производится потому, что только при ошибке старший разряд кода устанавливается в единицу. А поскольку тип NTSTATUS определен как тип LONG (знаковый), то значение с единичным старшим битом будет восприниматься как отрицательное.

Описание большинства из представленных ниже функций является неполным, но достаточным для выполнения данной лабораторной работы. Полное описание этих и других функций можно найти в разделе MSDN, посвященном использованию CNG (http://msdn.microsoft.com/en-us/library/windows/desktop/aa376210%28 v=vs.85%29.aspx).

BCryptOpenAlgorithmProvider

Данная функция загружает и инициализирует провайдер конкретного алгоритма. Ее прототип имеет вид:

NTSTATUS WINAPI BCryptOpenAlgorithmProvider(

BCRYPT\_ALG\_HANDLE \*phAlgorithm, LPCWSTR pszAlgId,

LPCWSTR pszImplementation, DWORD dwFlags);

Параметр phAlgorithm представляет собой указатель на переменную, принимающую возвращаемый дескриптор провайдера указанного алгоритма. Идентификатор запрашиваемого алгоритма задает второй параметр pszAlgId. Он представляет собой указатель на Unicode-строку с нуль-символом в конце, содержащую зарегистрированное имя одного из алгоритмов. Для стандартного CNG-провайдера «Microsoft Primitive Provider» в файле bcrypt.h определены строковые константы с именами зарегистрированных алгоритмов и соответствующие им символические константы, которые, как правило, и указываются в качестве второго параметра. При выполнении данной работы для шифрования будет использоваться алгоритм AES, а для хэширования – SHA-256. Поэтому из всех определенных идентификаторов нам понадобятся только два. Их описание в файле bcrypt.h приведено ниже:

#define BCRYPT\_AES\_ALGORITHM L"AES" #define BCRYPT\_SHA256\_ALGORITHM L"SHA256"

Параметр pszImplementation является указателем на Unicode-строку, содержащую зарегистрированное имя (псевдоним) одного из провайдеров криптографических примитивов. Если в качестве этого параметра указать константу NULL, то будет использован провайдер, определенный в реестре для данного алгоритма по умолчанию («Microsoft Primitive Provider»).

Параметр dwFlags для реализации обычного шифрования или хэширования должен быть равен нулю.

Ниже показан пример вызова функции для подключения к провайдеру алгоритма AES и один из возможных вариантов обработки возвращаемого функцией значения:

NTSTATUS status = STATUS\_UNSUCCESSFUL;

if(!BCRYPT\_SUCCESS(status = BCryptOpenAlgorithmProvider(

&hAlg, BCRYPT\_AES\_ALGORITHM, NULL, 0)))

{

\_tprintf(TEXT("Ошибка 0x%x получения\ дескриптора алгоритма"), status);

}

Если необходимо подключиться к провайдеру алгоритма хэширования SHA-256, то используется идентификатор BCRYPT\_SHA256\_ALGORITHM.

Полученный через параметр phAlgorithm дескриптор алгоритма, в зависимости от его типа, в дальнейшем используется для создания либо ключевого объекта, либо хэш-объекта.

BCryptGetProperty

Функция позволяет получить значение одного из свойств объекта CNG (алгоритма, ключевого объекта, хэш-объекта) и имеет следующий прототип:

NTSTATUS WINAPI BCryptGetProperty(

BCRYPT\_HANDLE hObject, LPCWSTR pszProperty, PUCHAR pbOutput,

ULONG cbOutput, ULONG \*pcbResult, ULONG dwFlags);

В качестве параметра hObject в функцию может передаваться дескриптор любого из объектов CNG, указанных выше. Параметр pszProperty является указателем на Unicode-строку, содержащую имя запрашиваемого свойства. Значения свойств могут быть разных типов, но все они возвращаются через параметр pbOutput. По сути функции передается адрес буфера в памяти, куда передается значение свойства. Поэтому передаваемый функции параметр должен приводиться к типу PUCHAR или PBYTE. Размер буфера задается параметром cbOutput, а параметр pcbResult задает адрес переменной, куда возвращается размер данных в байтах, реально скопированных в буфер pbOutput. Параметр dwFlags должен быть равен нулю.

Данная функция совместно с функцией BCryptSetProperty в качестве идентификаторов свойств может использовать символические константы, определенные в файле bcrypt.h. Некоторые из них приведены ниже:

 BCRYPT\_ALGORITHM\_NAME. Позволяет определить имя используемого алгоритма;

 BCRYPT\_BLOCK\_LENGTH. Позволяет получить размер блока алгоритма шифрования в байтах;

 BCRYPT\_CHAINING\_MODE. Позволяет получить название установленного режима шифрования. Для основных режимов блочного шифрования определены символические константы: BCRYPT\_CHAIN\_MODE\_CBC, BCRYPT\_CHAIN\_MODE\_CFB, BCRYPT\_CHAIN\_MODE\_ECB, которые соответствуют строковым константам, содержащим названия режимов;

 BCRYPT\_KEY\_LENGTH. Позволяет получить размер ключа в битах;

 BCRYPT\_HASH\_LENGTH. Позволяет получить размер хэш-кода загруженного алгоритма хэширования;

 BCRYPT\_OBJECT\_LENGTH. Используется для получения размера в байтах объектов провайдера некоторых типов.

Что касается последнего идентификатора, то он используется при вызове функции, в частности, если нужно определить размер ключевого объекта перед созданием сеансового ключа или размер хэш-объекта. В приведенном ниже примере определяется размер объекта, тип которого зависит от типа загруженного алгоритма. Если переменная hAlg содержит дескриптор алгоритма шифрования, то через параметр pbOutput возвратится размер ключевого объекта, а если загружен алгоритм хэширования, то вернется размер хэш-объекта. В приведенном ниже примере и во всех последующих для краткости не записывается «обвязка» вызова, проверяющая возвращаемое функцией значение.

DWORD cbObject=0, cbData=0; PBYTE pbObject=NULL;

BCryptGetProperty(hAlg, BCRYPT\_OBJECT\_LENGTH,

(PBYTE)&cbObject, sizeof(DWORD), &cbData, 0);

//выделение памяти для объекта:

pbObject=(PBYTE)HeapAlloc(GetProcessHeap(), 0,

cbObject);

//выполнение каких-либо операций с объектом

.............................................

//освобождение объекта

HeapFree(GetProcessHeap(), 0, pbObject);

Далее созданный в памяти буфер может быть передан функциям, которые разместят в нем, соответственно, или ключевой объект, или хэш-объект. В переменную cbData в этом случае будет записан размер в байтах переданных в cbObject данных, то есть 4.

BCryptSetProperty

Функция позволяет установить значение одного из именованных свойств объекта CNG (алгоритма, ключевого объекта, хэш-объекта). Ее прототип во многом повторяет прототип предыдущей функции:

NTSTATUS WINAPI BCryptSetProperty(

BCRYPT\_HANDLE hObject, LPCWSTR pszProperty, PUCHAR pbInput,

ULONG cbInput, ULONG dwFlags);

Разница между этой функцией и предыдущей заключается в том, что через параметр pbInput передается новое значение свойства, заданного параметром pszProperty. Через параметр cbInput передается размер этого значения. Параметр dwFlags также должен иметь значение нуля.

Формально функция BCryptSetProperty может использовать те же идентификаторы, что и функция BCryptGetProperty. Однако из всех перечисленных в описании предыдущей функции идентификаторов, функция BCryptSetProperty может использовать только один: BCRYPT\_CHAINING\_MODE. Другие рассмотренные идентификаторы определяют свойства, доступные только для чтения. Ниже показан пример вызова функции, устанавливающий для полученного ранее дескриптора алгоритма симметричного шифрования режим CFB:

BCryptSetProperty(hAlg, BCRYPT\_CHAINING\_MODE,

(PBYTE)BCRYPT\_CHAIN\_MODE\_CFB, sizeof(BCRYPT\_CHAIN\_MODE\_CFB), 0);

BCryptCreateHash

Функция создает хэш-объект и имеет следующий прототип:

NTSTATUS WINAPI BCryptCreateHash(

BCRYPT\_ALG\_HANDLE hAlgorithm, BCRYPT\_HASH\_HANDLE \*phHash, PUCHAR pbHashObject,

ULONG cbHashObject, PUCHAR pbSecret, ULONG cbSecret, ULONG dwFlags);

Параметр hAlgorithm представляет полученный ранее с помощью функции BCryptOpenAlgorithmProvider дескриптор алгоритма хэширования. Параметр phHash является указателем на переменную типа BCRYPT\_HASH\_HANDLE, которая будет содержать дескриптор

создаваемого хэш-объекта. В дальнейшем этот дескриптор будет передаваться функциям, которые непосредственно выполняют хэширование данных и выгрузку хэш-кода.

Параметр pbHashObject содержит указатель на буфер, в котором размещается созданный хэш-объект. Способ получения этого буфера был описан ранее. Параметр cbHashObject содержит размер этого буфера. Параметр pbSecret передает адрес буфера, содержащего ключ, используемый для создания кода аутентичности сообщения (MAC). Поскольку мы будем выполнять обычное хэширование, то вместо этого параметра нужно указывать константу NULL, а в качестве значения параметра cbSecret – 0. Параметр dwFlags в нашем случае тоже должен быть равен нулю.

Ниже показан пример вызова этой функции. Предполагается, что буфер, адресуемый указателем pbHashObject и его размер cbHashObject получены заранее:

BCryptCreateHash(hAlg, &hHash, pbHashObject,

cbHashObject, NULL, 0, 0);

BCryptHashData

Функция непосредственно выполняет хэширование передаваемых ей данных, используя созданный ранее с помощью функции BCryptCreateHash хэш-объект. Прототип функции имеет следующий вид:

NTSTATUS WINAPI BCryptHashData(

BCRYPT\_HASH\_HANDLE hHash,

PUCHAR pbInput, ULONG cbInput, ULONG dwFlags);

Параметр hHash представляет собой дескриптор существующего хэш-объекта. Параметр pbInput содержит адрес буфера с входными данными, а параметр cbInput его размер. Параметр dwFlags должен быть равен нулю. Функция может вызываться многократно для одного и того же хэш-объекта, добавляя в него новые данные для хэширования.

Ниже показан пример хэширования пароля, который находится в строке szPass.

TCHAR szPass[100];

..................................

//Получение каким-либо образом пароля и занесение его в szPass.

..................................

BCryptHashData(hHash,(PBYTE)szPass,

\_tcslen(szPass)\*sizeof(TCHAR),0);

BCryptFinishHash

Функция извлекает из хэш-объекта окончательный хэш-код входных данных, полученный с помощью одного или нескольких вызовов функции BCryptHashData. Прототип функции имеет вид:

NTSTATUS WINAPI BCryptFinishHash(

BCRYPT\_HASH\_HANDLE hHash,

PUCHAR pbOutput, ULONG cbOutput, ULONG dwFlags);

Параметр hHash является дескриптором хэш-объекта. Буфер, заданный указателем pbOutput, будет принимать выгружаемый хэш- код и должен быть создан заранее. Параметр cbOutput содержит размер этого буфера. Параметр dwFlags должен быть равен нулю.

Ниже показан пример выгрузки хэш-кода:

PBYTE pbHash= NULL;

DWORD cbData= 0, cbHash= 0; BCryptGetProperty(hAlg, BCRYPT\_HASH\_LENGTH,

(PBYTE)&cbHash, sizeof(DWORD), &cbData, 0);

pbHash= (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash); BCryptFinishHash(hHash, pbHash, cbHash, 0);

BCryptGenerateSymmetricKey

Функция создает ключевой объект и помещает в него ключ симметричного шифрования загруженного ранее алгоритма. Ключ создается на основе ключевого материала, передаваемого функции. Ее прототип имеет вид:

NTSTATUS WINAPI BCryptGenerateSymmetricKey(

BCRYPT\_ALG\_HANDLE hAlgorithm, BCRYPT\_KEY\_HANDLE \*phKey,

PUCHAR pbKeyObject, ULONG cbKeyObject, PUCHAR pbSecret, ULONG cbSecret, ULONG dwFlags);

Параметр hAlgorithm представляет полученный ранее с помощью функции BCryptOpenAlgorithmProvider дескриптор алгоритма шифрования. Параметр phKey является указателем на переменную типа BCRYPT\_KEY\_HANDLE, которая будет содержать дескриптор создаваемого ключевого объекта. Параметр pbKeyObject содержит указатель на буфер, в котором размещается созданный ключевой

объект. Способ получения этого буфера был описан ранее. Параметр

cbKeyObject содержит размер данного буфера в байтах.

Параметр pbSecret является указателем на буфер, который содержит ключевой материал. По сути, он и будет являться ключом. Получить ключевой материал можно двумя способами: генерацией псевдослучайного числа требуемого размера или созданием хэш-кода парольной фразы. Поэтому в нашем случае в качестве этого параметра функции будет передаваться указатель на буфер с хэш-кодом, извлеченным ранее функцией BCryptFinishHash. Через параметр cbSecret передается размер в байтах буфера с ключевым материалом.

Если вспомнить генерацию сеансового ключа AES в CryptoAPI, то там использовались отдельные идентификаторы алгоритма для разных длин ключа. Здесь же длина генерируемого ключа фактически определяется значением, передаваемым через параметр cbSecret. Если, к примеру, через параметр pbSecret передается адрес буфера, содержащего 32-байтный (256-битный) хэш-код, то задавая через параметр cbSecret значения 16, 24 или 32 мы получим ключ длиной, соответственно, 128, 192 или 256 бит.

Параметр dwFlags должен быть равен нулю. Ниже показан пример вызова функции для создания 128-битного ключа. Предполагается, что на момент вызова функции уже определены значения параметров pbKeyObject (содержит адрес буфера для размещения ключевого объекта), cbKeyObject (содержит размер буфера в байтах) и pbHash (содержит адрес буфера с хэш-кодом).

BCryptGenerateSymmetricKey(hAlg, &hKey, pbKeyObject,

cbKeyObject, (PBYTE)pbHash, 16, 0);

Для созданного после вызова функции дескриптора ключа можно определять и устанавливать некоторые параметры с помощью функций BCryptGetProperty и BCryptSetProperty. Например, если установлен режим шифрования CFB, то по умолчанию он используется в 8-битном варианте. Для того чтобы установить иной размер обратной связи (в байтах) можно использовать функцию BCryptSetProperty с идентификатором BCRYPT\_MESSAGE\_BLOCK\_LENGTH.

После создания ключа и установления его параметров (если это необходимо) можно передавать его дескриптор функциям зашифрования или расшифрования.

BCryptEncrypt

Функция зашифровывает переданный ей блок данных произвольного размера. Ее прототип имеет вид:

NTSTATUS WINAPI BCryptEncrypt(

BCRYPT\_KEY\_HANDLE hKey,

PUCHAR pbInput, ULONG cbInput,

VOID \*pPaddingInfo, PUCHAR pbIV,

ULONG cbIV,

PUCHAR pbOutput, ULONG cbOutput, ULONG \*pcbResult, ULONG dwFlags);

Через параметр hKey передается дескриптор созданного ранее ключа. Параметр pbInput содержит адрес буфера с входными данными (открытым текстом). Параметр cbInput содержит размер входных данных в байтах. Параметр pPaddingInfo может содержать указатель на одну из двух возможных структур, определяющих параметры дополнения блоков при шифровании. В случае обычного симметричного шифрования этот параметр не используется и должен быть равен NULL.

Через параметр pbIV можно передавать адрес буфера, содержащего вектор инициализации (синхропосылку). Обычно он получается с помощью генератора псевдослучайных чисел. Размер вектора в байтах передается через параметр cbIV и он должен совпадать с размером блока. То есть если размер блока алгоритма AES равен 16 байтам (128 бит), то вектор инициализации должен иметь такой же размер. Также нужно учесть, что в процессе зашифрования функция изменяет содержимое буфера с вектором. Поэтому если в дальнейшем вновь понадобится использовать его значение, то перед вызовом функции необходимо создать копию вектора. Если при зашифровании используется нулевой вектор инициализации, то параметры pbIV и cbIV должны быть заданы, соответственно, как NULL и 0.

Параметр pbOutput содержит адрес буфера, в который будет помещаться шифртекст. Размер буфера в байтах передается через параметр cbOutput. Если в качестве параметра pbOutput передать NULL, то через параметр pcbResult будет возвращен размер выходного буфера, необходимого для размещения шифртекста, который получится из указанных входных данных. Если же параметр pbOutput

при вызове содержит адрес выходного буфера, то через pcbResult

возвращается количество байт, реально помещенных в этот буфер.

Параметр dwFlags в случае симметричного шифрования может быть равен нулю или константе BCRYPT\_BLOCK\_PADDING, которая определяет необходимость дополнения последнего блока. Флаг устанавливается в ноль, если четко известно, что размер входных данных кратен размеру блока алгоритма симметричного шифрования. Если же при нулевом флаге размер открытого текста не кратен размеру блока, то восстановить его в дальнейшем может не получиться, вне зависимости от выбранного режима шифрования. Поэтому при шифровании файлов флаг всегда нужно устанавливать равным BCRYPT\_BLOCK\_PADDING, независимо от выбранного режима шифрования (в том числе и при выборе режима CFB). Следовательно, размер выходных данных будет всегда больше (для AES на величину от 1 до 16 байт), чем входных.

Отсюда следует еще один вывод. Параметры pbInput и pbOutput могут содержать адрес одного и того же буфера. Однако этот буфер нельзя полностью заполнять входными данными, так как заменяющие их по мере зашифрования выходные данные будут иметь больший размер (для AES максимально на 16 байт).

Ниже показан пример зашифрования массива однобайтных величин с нулевым вектором инициализации и предварительным определением размера буфера для размещения шифртекста:

BYTE Plaintext[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

DWORD cbOutput=0, cbCiphertext=0; PBYTE pbOutput=NULL;

BCryptEncrypt(hKey, Plaintext, sizeof(Plaintext), NULL, NULL, 0, NULL, 0, &cbOutput, BCRYPT\_BLOCK\_PADDING);

pbOutput=(PBYTE)HeapAlloc(GetProcessHeap(), 0,

cbOutput); BCryptEncrypt(hKey, Plaintext, sizeof(Plaintext),

NULL, NULL, 0, pbOutput, cbOutput, &cbCiphertext, BCRYPT\_BLOCK\_PADDING);

BCryptDecrypt

Функция расшифровывает переданный ей блок данных произвольного размера. Ее прототип имеет вид:

NTSTATUS WINAPI BCryptDecrypt(

BCRYPT\_KEY\_HANDLE hKey,

PUCHAR pbInput, ULONG cbInput,

VOID \*pPaddingInfo,

PUCHAR pbIV, ULONG cbIV,

PUCHAR pbOutput, ULONG cbOutput, ULONG \*pcbResult, ULONG dwFlags);

Все параметры этой функции по назначению аналогичны параметрам функции BCryptEncrypt. Отличие состоит лишь в том, что через параметр pbInput передается адрес буфера с шифртекстом, а параметр pbOutput указывает на буфер, в котором будет размещен восстановленный открытый текст. Рассмотренный ранее способ определения размера выходного буфера может применяться и для этой функции. Параметр dwFlags также должен иметь значение BCRYPT\_BLOCK\_PADDING.

BCryptDestroyHash

Функция уничтожает созданный ранее хэш-объект и имеет следующий прототип:

NTSTATUS WINAPI BCryptDestroyHash(

BCRYPT\_HASH\_HANDLE hHash);

Единственный параметр представляет собой дескриптор уничтожаемого хэш-объекта.

BCryptDestroyKey

Функция уничтожает созданный ранее ключевой объект и имеет следующий прототип:

NTSTATUS WINAPI BCryptDestroyKey(

BCRYPT\_KEY\_HANDLE hKey);

В качестве параметра функция принимает дескриптор уничтожаемого ключа.

BCryptCloseAlgorithmProvider

Функция выгружает загруженный ранее провайдер алгоритма и имеет следующий прототип:

NTSTATUS WINAPI BCryptCloseAlgorithmProvider(

BCRYPT\_ALG\_HANDLE hAlgorithm, ULONG dwFlags);

Параметрами функции являются дескриптор провайдера и флаг, который должен быть равен нулю.

СОДЕРЖАНИЕ РАБОТЫ

Разработать на языке программирования C/C++ консольное или оконное приложение, выполняющее зашифрование и расшифрование файла произвольного формата с помощью алгоритма AES-128, AES-192 или AES-256 по выбору пользователя. Программа должна генерировать сеансовый ключ на основании хэшированного пароля, который запрашивается у пользователя. Использовать хэш-функцию SHA-256.

Приложение должно по выбору пользователя использовать функционал двух криптографических интерфейсов:

 CryptoAPI (в этом случае режим блочного шифрования – устанавливаемый по умолчанию CBC с нулевым вектором инициализации, используемый криптопровайдер – «Microsoft Enhanced RSA and AES Cryptographic Provider»).

 Cryptography API: Next Generation (дополнительно предоставляется выбор режим блочного шифрования: ECB, CBC или CFB, причем в последних двух также используется нулевой вектор инициализации).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое CryptoAPI? В чем заключается различие между CryptoAPI 1.0 и CryptoAPI 2.0?

2. Что такое криптопровайдер? Как можно подключиться к криптопровайдеру?

3. Какое количество функций должен поддерживать криптопровайдер?

4. Как создать контейнер ключей? Какие типы ключей в нем будут храниться?

5. Какие типы криптопровайдеров вы знаете? Чем они различаются?

6. Как можно выполнить генерацию ключа симметричного шифрования?

7. Какой режим шифрования устанавливается при генерации ключа по умолчанию?

8. Что такое хэш-объект? Какие функции для работы с хэш- объектами вы знаете?

9. Какие функции CryptoAPI выполняют зашифрование и расшифрование данных? Какие они имеют параметры?

10. Что такое Cryptography API: Next Generation? В чем заключаются его различия с CryptoAPI?

11. Какие типы провайдеров CNG доступны в операционных системах Windows? Как можно узнать, какие конкретно провайдеры установлены в системе?

12. Средства каких провайдеров CNG можно использовать в режиме ядра?

13. Какие алгоритмы поддерживает провайдер криптографических примитивов Microsoft?

14. Что такое маршрутизатор примитивов?

15. Как определить успешность вызова функции CNG?

16. Как загрузить и выгрузить провайдер нужного алгоритма?

17. Как создается хэш-объект? Какие функции CNG для работы с хэш-объектами вы знаете?

18. Как определить размер буфера при выгрузке хэш-кода?

19. Как сгенерировать ключ симметричного шифрования и установить его параметры?

20. Какие функции CNG выполняют зашифрование и расшифрование данных? Какие они имеют параметры?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 10 СИММЕТРИЧНОЕ И АСИММЕТРИЧНОЕ ШИФРОВАНИЕ ДАННЫХ СРЕДСТВАМИ КРИПТОГРАФИЧЕСКОГО ПАКЕТА OPENSSL**

Цель работы: ознакомиться со средствами симметричного и асимметричного шифрования, предоставляемыми пакетом OpenSSL и способами доступа к ним из приложений на языке C/C++.

ОСНОВНЫЕ ПОНЯТИЯ

Криптографический пакет OpenSSL

Пакет предназначен прежде всего для обеспечения работы веб- серверов, поддерживающих передачу данных через защищенные протоколы SSL (Secure Sockets Layer) и TLS (Transport Layer Security). В частности, он позволяет генерировать ключевые пары различных асимметричных алгоритмов, сеансовые ключи блочных шифров, выполнять шифрование, создавать электронные подписи и т.д.

Взаимодействие с OpenSSL возможно в двух вариантах: через командную строку в режиме консольного приложения или непосредственное использование в приложении функций динамических библиотек, поставляемых в составе пакета. В этой работе будем использовать второй вариант, непосредственно вызывая нужные нам функции в тексте программы на языке C/C++. Далее вкратце опишем процесс инсталляции и первоначальной настройки пакета, а также порядок использования его библиотек в приложении Win32, создаваемом в среде Visual Studio.

Как правило, OpenSSL используется совместно с Unix-подобными системами, но как кроссплатформенное приложение может применяться и в ОС Windows. Документация, а также последняя версия пакета доступны по адресу: https://www.openssl.org. Загруженный пакет необходимо скомпилировать и сконфигурировать для работы в конкретной ОС. Для того чтобы избавить пользователя Windows от выполнения большинства этих действий существуют готовые дистрибутивы для 32 и 64-разрядных версий системы, доступные по адресу: http://slproweb.com/products/Win32OpenSSL.html. Кроме дистрибутива OpenSSL, необходимо с этой же страницы скачать и предварительно установить в системе пакет Visual C++ 2008

Redistributables (соответственно разрядности ОС). После его установки, необходимо запустить дистрибутив самого OpenSSL и далее идет процесс обычной инсталляции Windows-приложения с указанием пользователем места расположения копируемых файлов. Поскольку в среде Visual Studio будет создаваться 32-разрядное приложение, то рекомендуется скачивать и устанавливать 32- разрядную версию пакета OpenSSL независимо от разрядности ОС, так как в противном случае разрядность приложения и установленных динамических библиотек будет различаться и проект не соберется.

После установки пакет в принципе готов к работе, однако необходимо осуществить еще некоторые действия по его конфигурации. При рассмотрении всех примеров будем считать, что пакет был установлен в каталоге C:\OpenSSL-Win32. Непосредственно для работы с командной строки используется файл openssl.exe, расположенный в подкаталоге bin. Для проверки работоспособности пакета запустим этот файл на выполнение.

В консоли появится предупреждение и приглашение для ввода команд:

WARNING: can't open config file: /usr/local/ssl/openssl.cfg OpenSSL>

Предупреждение означает, что при запуске не найден конфигурационный файл openssl.cfg. Приложение без этого файла остается работоспособным, что подтверждается выдачей приглашения. Однако в этом файле указывается ряд параметров, которые могут понадобиться в процессе эксплуатации пакета. Например, в нем указываются параметры подключения модуля gost, который реализует отечественные алгоритмы шифрования, хэширования и электронной подписи. Модуль с реализацией отечественных стандартов криптографии, появившийся в версии 1.0.0, был разработан компанией

«Криптоком». Такой модуль присутствует по умолчанию в дистрибутивах, скачанных на ресурсе http://slproweb.com. Для его включения пока завершим работу OpenSSL, введя команду exit (или quit).

С помощью Проводника откроем каталог, где по умолчанию находится конфигурационный файл – C:\OpenSSL-Win32\bin. С помощью любого текстового редактора откроем файл openssl.cfg. Этот файл делится на ряд секций, названия которых указываются в квадратных скобках. Пока не будем вносить в них изменений, а добавим строки, включающие поддержку алгоритмов ГОСТ.

В качестве первой строки файла добавим указание на первую секцию настройки параметров подключения модуля:

openssl\_conf = openssl\_def

Секцию [ openssl\_def ] и другие секции настройки параметров подключения модуля gost разместим в конце файла:

[openssl\_def] engines=engine\_section

[engine\_section] gost=gost\_section

[gost\_section] engine\_id=gost

dynamic\_path = C:/OpenSSL-Win32/bin/gost.dll default\_algorithms=ALL

CRYPT\_PARAMS=id-Gost28147-89-CryptoPro-A-ParamSet

В последней секции указывается имя модуля, по которому к нему в дальнейшем можно обращаться в программе (gost), путь к dll-файлу модуля gost, включаемые алгоритмы (все реализованные) и параметр, являющийся идентификатором варианта таблицы замен алгоритма ГОСТ 28147-89, представленном в документе RFC 4357.

Выполнив все изменения, сохраним файл и закроем его. Теперь можно создать переменную среды, которая укажет для пакета путь доступа к конфигурационному файлу. Ее можно создать средствами Проводника (Свойства системы  Дополнительные параметры системы  Переменные среды) или командой в окне командной строки:

set OPENSSL\_CONF=c:\openssl-win32\bin\openssl.cfg

Вновь запустим на выполнение файл openssl.exe и убедимся, что предупреждение не появляется. Проверим, появилась ли поддержка отечественных криптоалгоритмов. Для этого после приглашения введем команду:

ciphers -v

и убедимся, что среди представленных наборов алгоритмов (шифр- сьютов) есть такие, в которых фигурируют названия GOST-89, GOST-94, GOST2001 (соответственно ГОСТ 28147-89, ГОСТ Р 34.10-94, ГОСТ Р 34.10-2001). Закроем окно консоли.

Настроенная загрузка алгоритмов ГОСТ понадобится позже, а пока рассмотрим, как производится симметричное и асимметричное

шифрование средствами стандартного модуля пакета при доступе к его функциям из программы на языке C/C++.

В стандартную реализацию пакета входит большое число симметричных шифров (с возможностью работы в различных режимах) и несколько асимметричных алгоритмов шифрования, обмена ключами и электронной подписи (ЭП). В данной работе ограничимся использованием алгоритма AES для симметричного шифрования и RSA для асимметричного (имитации процесса обмен сеансовым ключом). Далее будут рассмотрен минимально необходимый для выполнения данной работы набор функций.

Инициализация библиотеки шифрования

Для начала посмотрим на содержимое каталога, в который установлен пакет OpenSSL. Из того, что в нем есть, нам понадобится:

 dll-файлы динамических библиотек, которые находятся в подкаталоге bin. В данной работе будут использоваться функции, входящие в состав библиотек libeay32.dll и gost.dll.

 lib-файлы статических библиотек (используются как библиотеки импорта подключаемых dll-библиотек), которые находятся в подкаталоге lib. В этой работе нам понадобятся библиотеки libeay32.lib и ssleay32.lib. Их необходимо подключить, например в свойствах проекта VC++ (Проект  Свойства  Свойства конфигурации  Компоновщик  Ввод  Дополнительные зависимости).

 Заголовочные файлы, содержащие прототипы функций, типы данных, константы, и которые находятся в подкаталоге include. Необходимые заголовочные файлы включаются в текст программы на языке С/С++ с помощью директивы препроцессора #include. В настройках проекта также нужно добавить каталог C:\OpenSSL-Win32\include (в нашем случае) в раздел каталогов VC++ (Проект  Свойства  Свойства конфигурации  Каталоги VC++  Каталоги включения).

В принципе, вызывать большинство функций, зная их сигнатуру (если подключить заранее нужный заголовочный файл, то ее подскажет сама IDE) можно без какой-то особой инициализации. Однако мы в качестве этой процедуры выполним в начале программы вызов двух функций, описанных ниже.

Функция OpenSSL\_add\_all\_algorithms загружает во внутренние таблицы библиотеки все имеющиеся в стандартном модуле реализации

алгоритмы шифрования и хэширования. Прототип функции содержится в файле openssl/evp.h и имеет вид:

void OpenSSL\_add\_all\_algorithms(void);

Если в процессе выполнения функций OpenSSL проиcходит ошибка, то для просмотра ее текстового описания нужно, чтобы была инициализирована внутренняя таблица сообщений об ошибках. Это делается с помощью функции ERR\_load\_error\_strings. Она объявлена в файле openssl/err.h:

void ERR\_load\_crypto\_strings(void);

Далее рассмотрим, какие средства ввода-вывода библиотеки OpenSSL нам понадобятся в процессе выгрузки информации в файл и ее загрузки из файла.

Абстракция ввода-вывода BIO

В OpenSSL имеется специальный тип для представления разных видов источников и получателей данных – BIO. Он скрывает от пользователя детали операций ввода-вывода, позволяя использовать для их реализации некоторый набор функций. Имеется два основных типа объектов BIO: источник/получатель данных или фильтр (конструкция, позволяющая связывать несколько структур BIO и осуществлять некоторые преобразования данных при передаче их из одной структуры в другую). Источник или получатель данных могут представлять собой файл, сокет или просто буфер в оперативной памяти. Независимо от того, к какому типу будет принадлежать создаваемый объект BIO, в программе переменная для работы с ним задается как указатель на одноименный тип BIO, определенный в файле openssl/bio.h. Тип BIO получен переименованием структурного типа bio\_st, определенного в этом же файле и содержащего ряд полей, задающих информацию об объекте. Содержание структуры можно посмотреть в заголовочном файле.

В данной работе мы будем использовать только файловый BIO. Он создан на базе типа FILE стандартной библиотеки ввода-вывода языка Си. Создать переменную такого типа можно с помощью функции BIO\_new\_file, объявленную в файле openssl/bio.h.

BIO \*BIO\_new\_file(const char \*filename, const char \*mode);

Параметры функции аналогичны таковым у функции fopen.

Пример вызова:

BIO \*fbio;

fbio = BIO\_new\_file("outfile.txt", "wb");

Для выполнения операций чтения-записи данных в файловом BIO

имеется ряд функций, из которых мы рассмотрим только две.

BIO\_read пытается прочитать заданное число байт структуры BIO. Имеет прототип:

int BIO\_read(BIO \*b, void \*data, int len);

Параметр len определяет количество байт, которые функция пытается прочесть из объекта b и поместить их в буфер data. Функция возвращает количество реально считанных байт данных.

BIO\_write пытается записать len байт буфера data в b. Имеет прототип:

int BIO\_write(BIO \*b, const void \*data, int len);

Для удаления объекта BIO, независимо от его типа, используется функция BIO\_free.

void BIO\_free(BIO \*b);

С другими функциями для работы с объектами BIO можно ознакомиться в документации к пакету OpenSSL.

Генерация псевдослучайных чисел

В симметричном шифровании псевдослучайные числа используются для генерации секретного ключа и вектора инициализации (синхропосылки) для режимов, отличных от ECB. Для генерации псевдослучайных чисел используются две функции: RAND\_bytes и RAND\_pseudo\_bytes, прототипы которых объявлены в файле openssl/rand.h.

int RAND\_bytes(unsigned char \*buf, int num);

int RAND\_pseudo\_bytes(unsigned char \*buf, int num);

Первая функция генерирует num криптографически сильных псевдослучайных чисел в буфер buf. Результат ее работы можно использовать в качестве сеансовых ключей симметричных криптоалгоритмов. Функция возвращает 1 в случае успеха и 0 в случае неудачи.

Функция RAND\_pseudo\_bytes действует аналогично, но достаточные статистические характеристики чисел не гарантируется. В случае если полученные псевдослучайные данные криптографически сильные, возвращает 1, в случае если они недостаточно сильные, 0 и -1 в случае ошибки. Результат работы данной функции можно использовать в качестве вектора инициализации.

Однако перед тем как генерировать псевдослучайные числа, желательно инициализировать генератор, внеся в его параметры источник энтропии (seed). Простейшим вариантом может быть вызов функции RAND\_screen, которая берет случайные значения из хэша данных, полученных из скриншота содержимого экрана.

Более предпочтительным вариантом является непосредственное указание случайных данных с помощью функции RAND\_add.

void RAND\_add(const void \*buf, int num, double entropy);

Случайные величины в количестве num байт берутся из буфера buf. Параметр entropy задает энтропию сообщения, содержащегося в буфере. В общем случае можно задавать этот параметр равным num.

С другими функциями, объявленными в файле openssl/rand.h

можно ознакомиться в документации к библиотеке.

Симметричное шифрование данных с помощью криптоалгоритма AES

Для симметричного шифрования будем использовать высокоуровневые функции, имеющие в своем названии префикс

«EVP» и объявленные в файле openssl/evp.h.

Независимо от того, какой конкретно блочный шифр применяется, используется одинаковый набор функций для инициализации процесса шифрования, загрузки данных и завершения процесса. Эти функции используют две структуры, определенные в этом же заголовочном файле: EVP\_CIPHER и EVP\_CIPHER\_CTX.

Первая структура содержит информацию о блочном шифре: размер блока, длина ключа и т.п., а также ряд указателей на функции, которые нужно инициализировать. Для этого существует несколько способов. Мы будем использовать непосредственный вызов функций, возвращающих константный указатель на динамически созданную структуру EVP\_CIPHER, в названии которых содержатся данные об алгоритме, длине ключа и режиме шифрования. В данной работе ограничимся шифрованием AES со 128-разрядным ключом в режимах CBC и 128-разрядный CFB (размер блока в любых режимах всегда равен 128 бит). Соответственно это будут функции:

const EVP\_CIPHER \*EVP\_aes\_128\_cbc(void);

const EVP\_CIPHER \*EVP\_aes\_128\_cfb128(void);

Структура EVP\_CIPHER\_CTX представляет собой контекст алгоритма шифрования, первоначальную инициализацию которого осуществляют с помощью функции EVP\_CIPHER\_CTX\_init.

void EVP\_CIPHER\_CTX\_init(EVP\_CIPHER\_CTX \*a);

Данная функция вызывается, если переменная типа EVP\_CIPHER\_CTX описана статически. Если она описана как указатель, то вызывается функция EVP\_CIPHER\_CTX\_new, которая выделяет под нее память и возвращает адрес структуры.

EVP\_CIPHER\_CTX \*EVP\_CIPHER\_CTX\_new();

В дальнейшем освобождает контекст алгоритма шифрования либо функция:

int EVP\_CIPHER\_CTX\_cleanup(EVP\_CIPHER\_CTX \*a);

в случае статического размещения структуры, либо функция:

void EVP\_CIPHER\_CTX\_free(EVP\_CIPHER\_CTX \*a);

После того, как инициализирован контекст алгоритма, можно вызвать функцию EVP\_EncryptInit, которая подготавливает его к выполнению операций шифрования:

int EVP\_EncryptInit(EVP\_CIPHER\_CTX \*ctx,

const EVP\_CIPHER \*cipher, const unsigned char \*key, const unsigned char \*iv);

Параметры key и iv представляют собой буферы, содержащие ключевой материал и вектор инициализации, которые получаются с помощью функций генерации псевдослучайных чисел. Функция, как и большинство других, имеющих тип int, возвращает 1 в случае успеха и

0 в случае неудачи. Если вызов закончился неудачно, то можно вывести текстовое сообщение об ошибке. Ниже будет показан пример инициализации контекста, где вариант с ошибкой обрабатывается подобным образом. Такой подход можно использовать и при вызове других функций.

unsigned char keybuf[16] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,

10, 11, 12, 13, 14, 15 };

unsigned char iv[16] = { 0 }, EVP\_CIPHER\_CTX ctx;

EVP\_CIPHER\_CTX\_init(&ctx);

int ret = EVP\_EncryptInit(&ctx, EVP\_aes\_128\_cbc(), keybuf,

iv);

if (!ret)

{

char buffer[500]; ERR\_error\_string(ERR\_get\_error(), buffer); printf("%s\n",buffer);

}

В этом примере значения ключа и вектора инициализации для простоты заданы непосредственно, но на практике для их получения нужно использовать псевдослучайные числа. Для получения кода ошибки из очереди была использована функция ERR\_get\_error, а для получения текстового описания функция ERR\_error\_string. Прототипы этих функций описаны в файле openssl/err.h.

Для зашифрования открытого текста используются две функции:

int EVP\_EncryptUpdate(EVP\_CIPHER\_CTX \*ctx,

unsigned char \*out,

int \*outl, const unsigned char \*in, int inl);

и

int EVP\_EncryptFinal(EVP\_CIPHER\_CTX \*ctx,

unsigned char \*out, int \*outl);

Параметр ctx является адресом структуры контекста алгоритма. Параметры out и in являются, соответственно, выходным и входным буферами, а outl и inl их длинами (параметр outl в процессе работы функции может изменяться). Функции возвращают 1, если шифрование завершено успешно и 0 в противном случае.

Функция EVP\_EncryptUpdate зашифровывает данные порциями, размер которых может быть больше размера блока. Обычно все же длину буфера делают кратной длине блока. Если используется режим шифрования CBC, то он требует дополнения последнего блока. По умолчанию, режим создания дополнения включен. За его включение/отключение отвечает функция:

int EVP\_CIPHER\_CTX\_set\_padding(EVP\_CIPHER\_CTX \*c, int pad);

Если при ее вызове в качестве параметра pad задать 1, то режим создания дополнения будет включен (по умолчанию), а если 0, то выключен.

Функция EVP\_EncryptUpdate в режиме включенного дополнения зашифровывает порции данных размером, кратным длине блока (в

нашем случае 16 байт). Если последний фрагмент имеет длину не кратную длине блока, то функция EVP\_EncryptUpdate зашифрует максимально возможную его часть с длиной, кратной длине блока. Для шифрования оставшихся данных вызывается функция EVP\_EncryptFinal. Эта же функция вызывается и в том случае, если последний фрагмент имеет длину, кратную длине блока и нужно добавить еще один блок.

Если же используется режим шифрования, не требующий дополнения неполных блоков, например CFB, то режим создания дополнения отключается. В этом случае функции EVP\_EncryptUpdate на вход подают фрагменты данных любого размера и на выходе получают шифртекст такого же размера. А вызов функции EVP\_EncryptFinal не требуется.

При расшифровании данных, с контекстом алгоритма проводят те же действия по первоначальной инициализации, но для подготовки контекста используют функцию EVP\_DecryptInit. Функция имеет ту же сигнатуру, что и EVP\_EncryptInit. Естественно, алгоритм, режим шифрования, значения ключа и вектора инициализации должны быть теми же самыми, что и при зашифровании.

Непосредственно для расшифрования данных используются функции EVP\_DecryptUpdate и EVP\_DecryptFinal. Сигнатуры этих функций совпадают с сигнатурами функций, использовавшихся для зашифрования данных. Есть некоторый нюанс, связанный с расшифрованием последнего блока зашифрованных данных в режиме CBC. Для корректного расшифрования данных режим дополнения с помощью функции EVP\_CIPHER\_CTX\_set\_padding должен быть отключен для всех блоков, кроме последнего. Что касается режима CFB, то также как и при зашифровании, режим дополнения отключен постоянно.

В данной работе предполагается, что симметричное шифрование в программе будет проводиться с файлом, выбранным пользователем. Для доступа к файлу можно использовать обычные функции файлового ввода-вывода библиотеки языка C/C++, функции Win32 API или рассмотренный выше тип файлового BIO. Что касается шифрования самого сгенерированного сеансового ключа с целью его дальнейшего обмена, то оно должно проводиться средствами асимметричных криптоалгоритмов, например RSA.

Асимметричное шифрование данных с помощью криптоалгоритма RSA

Для реализации технологии создания «цифрового конверта» (то есть зашифрования сеансового ключа алгоритмом с открытым ключом), будем использовать алгоритм RSA. Структуры данных и прототипы функций, отвечающие за генерацию ключевых пар, шифрование и т.д. хранятся в файле openssl/rsa.h.

Для хранения в памяти информации о ключевой паре используется структура RSA. Структура создается динамически с помощью функции RSA\_new:

RSA \*RSA\_new(void);

При ее вызове указателю на структуру RSA присваивается возвращаемое значение.

Для генерации ключевой пары можно использовать функцию

RSA\_generate\_key:

RSA \*RSA\_generate\_key(int bits, unsigned long e,

void (\*callback) (int, int, void \*), void \*cb\_arg);

Параметр bits задает размер ключа в битах. По соображениям криптостойкости его значение должно быть не менее 2048 бит. Следующий параметр задает значение открытой экспоненты. Обычно в качестве e задают константу RSA\_F4 (0x10001L). Параметр callback является указателем на функцию обратного вызова, которая должна демонстрировать ход генерации ключевой пары. Последний параметр также используется данной функцией. Если никакой демонстрации хода вычислений не требуется, то последние два параметра можно задавать как NULL.

После генерации ключевой пары можно осуществить ее проверку на пригодность с помощью функции RSA\_check\_key:

int RSA\_check\_key(const RSA \*);

Функция вернет одно из трех значений: 1, если тест пройден успешно, 0, если ключевая пара тест не прошла и -1, если произошла ошибка при выполнении теста.

Для выполнения операции зашифрования данных с помощью открытого ключа используется функция RSA\_public\_encrypt:

int RSA\_public\_encrypt(int flen, const unsigned char \*from,

unsigned char \*to, RSA \*rsa, int padding);

Параметр flen является размером шифруемых данных. В нашем случае, если ключ алгоритма AES имеет размер 128 бит, то параметр flen должен быть равен 16. Следующий параметр определяет буфер с шифруемыми данными (т.е. с симметричным ключом AES). Параметр to задает адрес буфера, куда попадают зашифрованные данные. Размер этого буфера определятся предварительным вызовом функции RSA\_size:

int RSA\_size(const RSA \*rsa);

Если длина ключа алгоритма RSA (числа n) была задана как 2048 бит, то функция RSA\_size вернет значение 256 байт. Параметр padding задает способ дополнения. Его рекомендуют задавать равным константе RSA\_PKCS1\_OAEP\_PADDING. Длина входных данных при этом должна быть на 41 байт меньше размера выходного буфера. В нашем случае это требование выполняется с большим запасом.

Для расшифрования используется функция:

int RSA\_private\_decrypt(int flen, const unsigned char

\*from,

unsigned char \*to, RSA \*rsa, int padding);

Ее параметры аналогичны по назначению параметрам функции

RSA\_public\_encrypt. Параметр padding должен иметь то же значение.

Освобождение памяти, занятой структурой RSA, осуществляется функцией:

void RSA\_free(RSA \*r);

Для того чтобы сохранить сгенерированную ключевую пару в файле и извлечь ее при необходимости, можно использовать два вида функций. Одни выгружают пару в двоичном формате DER (Distinguished Encoding Rules), другие в формате PEM (Privacy Enhanced Mail), который представляет собой DER-формат, закодированный кодировкой base64. Далее будут рассмотрены функции для работы с форматом PEM, прототипы которых определены в файле openssl/pem.h.

Открытый и закрытый ключи записываются в разные файлы. Для записи открытого ключа применяется функция:

int PEM\_write\_bio\_RSAPublicKey(BIO \*bp, RSA \*x);

В качестве параметра bp можно использовать файловый BIO, открытый для записи. Функция возвращает 1 в случае удачного завершения и 0 в противном случае.

Закрытый ключ, как правило, хранится в зашифрованном виде. Он шифруется с помощью блочного шифра с ключом, получаемым из хэшированного пароля. Для выгрузки закрытого ключа может использоваться функция:

int PEM\_write\_bio\_RSAPrivateKey(BIO \*bp, RSA \*x,

const EVP\_CIPHER \*enc, unsigned char \*kstr, int klen, pem\_password\_cb \*cb, void \*u);

Параметр enc представляет собой константный указатель на структуру EVP\_CIPHER, вместо которого можно произвести вызов функции нужного алгоритма, например, EVP\_aes\_128\_cfb128().

Можно указать в параметре cb адрес функции обратного вызова, которая будет запрашивать пароль (ее пример есть в документации). Если указатель kstr не будет равен NULL, то в качестве пароля будут использованы первые klen символов из массива, на который указывает kstr, при этом параметр cb игнорируется. Если cb равен NULL, а параметр u не равен NULL, то u интерпретируется как строка, заканчивающаяся нулем, и эта строка используется как пароль. Также можно все параметры (kstr, cb, u) установить в NULL, и библиотека запросит у пользователя ввод парольной фразы (более 3-х символов) в консоли (причем с отключенным эхо-выводом).

Для извлечения закрытого ключа из файла применяется функция:

RSA \*PEM\_read\_bio\_RSAPrivateKey(BIO \*bp, RSA \*\*x,

pem\_password\_cb \*cb, void \*u);

Функция в качестве параметра x получает адрес указателя на созданную структуру RSA или NULL, если ее необходимо создать. Остальные параметры аналогичны функции PEM\_write\_bio\_RSAPrivateKey. Функция возвращает указатель на структуру с извлеченным ключом или NULL в случае ошибки.

Для извлечения открытого ключа используется функция:

RSA \*PEM\_read\_bio\_RSAPublicKey(BIO \*bp, RSA \*\*x,

pem\_password\_cb \*cb, void \*u);

Ее параметры аналогичны предыдущей функции. Причем последние два параметра всегда задаются равными NULL, так как при выгрузке в файл открытого ключа шифрование не производится. Функция также возвращает указатель на структуру с извлеченным ключом или NULL в случае ошибки.

Симметричное шифрование данных с помощью криптоалгоритма ГОСТ 28147-89

Ранее рассказывалось, как сконфигурировать пакет OpenSSL для использования отечественных симметричных и асимметричных криптоалгоритмов, реализация которых была добавлена в версии 1.0.0 компанией «Криптоком». В дальнейшем описании будем исходить из того, что конфигурационный файл дополнен информацией, позволяющей загрузить модуль gost и создана переменная среды OPENSSL\_CONF, содержащая путь к этому конфигурационному файлу.

Реализация отечественных криптоалгоритмов в OpenSSL осуществляется с использованием технологии дополнительных подгружаемых модулей (engine). Поэтому перед началом работы с модулем, реализующим поддержку алгоритмов ГОСТ, его необходимо активировать. Для этого существует несколько вариантов. Самым простым является считывание конфигурационного файла, в котором заранее определяются параметры загрузки модуля gost (как было показано выше). Для этого можно использовать функцию OPENSSL\_config, прототип которой определен в заголовочном файле openssl/conf.h.

Ранее предлагался вариант инициализации OpenSSL путем вызова двух функций: OpenSSL\_add\_all\_algorithms и ERR\_load\_error\_strings. Если предполагается считывание конфигурационного файла, путь к которому определен в переменной среды OPENSSL\_CONF, то эту операцию можно совместить с вызовом функции OpenSSL\_add\_all\_algorithms. Дело в том, что в действительности это макрос, который в зависимости от того, определена ли символическая константа OPENSSL\_LOAD\_CONF, замещается вызовом одной из двух функций:

void OPENSSL\_add\_all\_algorithms\_noconf(void); void OPENSSL\_add\_all\_algorithms\_conf(void);

Как правило, константа не определяется и вызывается первый вариант, который приводит только к загрузке алгоритмов. Если вместо макроса OpenSSL\_add\_all\_algorithms вызвать непосредственно функцию OPENSSL\_add\_all\_algorithms\_conf, то это позволит осуществить не только загрузку алгоритмов, но и считывание стандартного конфигурационного файла.

В целом для шифрования с помощью ГОСТ 28147-89 используются те же функции с префиксом EVP, что и при шифровании алгоритмом

AES. Небольшое отличие заключается в процессе настройки контекста алгоритма для выполнения зашифрования или расшифрования. Рассмотренные ранее функции EVP\_EncryptInit и EVP\_DecryptInit, выполнявшие эти операции, данном случае использовать нельзя, так как они ориентированы на вызов стандартного модуля реализации криптоалгоритмов. Поэтому для работы с модулем gost будем использовать их расширенные варианты: EVP\_EncryptInit\_ex и EVP\_DecryptInit\_ex (подробнее будут рассмотрены ниже), которые содержат дополнительный параметр, определяющий модуль реализации алгоритма. Он представляет собой указатель на структуру ENGINE, который можно предварительно получить с помощью функции ENGINE\_by\_id.

ENGINE \*ENGINE\_by\_id(const char \*id);

Объявления структуры и функции находятся в файле openssl/engine.h. Единственным параметром функции является константный указатель на строку id, содержащую имя модуля (engine). С учетом имени, заданного нами в конфигурационном файле, функцию можно вызвать так:

ENGINE \*engine\_gost = ENGINE\_by\_id("gost");

В случае ошибки, функция возвращает значение NULL.

Порядок действий, производимых для зашифрования или расшифрования, по сравнению с применением криптоалгоритма AES, в целом не меняется. Также в функциях используются структуры EVP\_CIPHER и EVP\_CIPHER\_CTX. При шифровании криптоалгоритмом AES можно было использовать специализированные функции типа EVP\_aes\_128\_cbc, которые возвращали константный указатель на структуру EVP\_CIPHER. Для алгоритмов ГОСТ таких специальных функций нет, поэтому используется более общий способ получения константного указателя на структуру EVP\_CIPHER. В частности, его можно получить, используя заданное имя алгоритма или численный идентификатор с помощью одной из функций:

const EVP\_CIPHER \*EVP\_get\_cipherbyname(const char \*name); const EVP\_CIPHER \*EVP\_get\_cipherbynid(int nid);

В качестве параметров данных функций можно использовать символические константы SN\_id\_Gost28147\_89 и NID\_id\_Gost28147\_89 соответственно, объявленные в файле openssl/obj\_mac.h (включать его в текст программы отдельно не нужно). Первая из констант замещается препроцессором на строку

"gost89", которую также непосредственно можно указывать в качестве параметра функции EVP\_get\_cipherbyname.

В созданной при вызове одной из двух описанных выше функций структуре EVP\_CIPHER содержатся параметры алгоритма, в том числе режим шифрования. По умолчанию устанавливается режим CFB (гаммирование с обратной связью). В этом можно убедиться, вызвав функцию:

int EVP\_CIPHER\_mode(const EVP\_CIPHER \*e);

Она вернет текущий режим в виде одной из предопределенных констант. Режиму CFB соответствует константа EVP\_CIPH\_CFB\_MODE (со значением 3). Что касается режима CBC, то, как известно, в стандарте ГОСТ 28147-89 он для шифрования не используется и, следовательно, в данной реализации отсутствует. Программист при необходимости может разработать собственную реализацию данного режима, если это потребуется. Мы же в данной работе ограничимся использованием устанавливаемого по умолчанию режима CFB.

Со структурой типа EVP\_CIPHER\_CTX производятся абсолютно те же действия, что и при шифровании алгоритмом AES. Сначала происходит инициализация контекста шифрования с помощью функции EVP\_CIPHER\_CTX\_init (при статическом размещении переменной) или EVP\_CIPHER\_CTX\_new (в случае динамической переменной). Далее контекст алгоритма подготавливается к выполнению операции зашифрования с помощью вызова уже упомянутой ранее функции:

int EVP\_EncryptInit\_ex(EVP\_CIPHER\_CTX \*ctx,

const EVP\_CIPHER \*cipher, ENGINE \*impl,

const unsigned char \*key, const unsigned char \*iv);

Единственным отличием сигнатуры данной функции от функции EVP\_EncryptInit является параметр impl, представляющий собой указатель на структуру ENGINE, предварительно получаемый с помощью функции ENGINE\_by\_id. Ниже показан пример инициализации и установки параметров контекста шифрования:

unsigned keybuf[8] = { 1, 2, 3, 4, 5, 6, 7, 8 };

unsigned char iv[8] = { 1, 2, 3, 4, 5, 6, 7, 8 }; ENGINE \*engine\_gost = ENGINE\_by\_id("gost"); EVP\_CIPHER\_CTX ctx;

EVP\_CIPHER\_CTX\_init(&ctx);

int ret = EVP\_EncryptInit\_ex(&ctx, EVP\_get\_cipherbynid(NID\_id\_Gost28147\_89), engine\_gost, (unsigned char\*) keybuf, iv);

Для простоты здесь не указывается обработка возвращаемого функцией EVP\_EncryptInit\_ex, а ключ и синхропосылка заданы непосредственно. В реальности же необходимо формировать их псевдослучайно. После вызова функции EVP\_EncryptInit\_ex можно осуществлять зашифрование открытого текста функцией EVP\_EncryptUpdate так же, как это делалось при использовании алгоритма AES. Поскольку установлен режим шифрования CFB, то вызов функции EVP\_EncryptFinal не требуется. По окончании зашифрования контекст алгоритма также освобождается вызовом функций EVP\_CIPHER\_CTX\_cleanup или EVP\_CIPHER\_CTX\_free.

Подготовка контекста алгоритма к расшифрованию шифртекста аналогична, только производится вызов функции EVP\_DecryptInit\_ex, имеющей сигнатуру, аналогичную функции EVP\_EncryptInit\_ex. Расшифрование блоков шифртекста также производится последовательным вызовом функции EVP\_DecryptUpdate.

Среди отечественных криптоалгоритмов нет такого, который можно было бы использовать непосредственно для реализации процедуры обмена сеансовым ключом, подобно RSA. В отечественной криптографии реализованы только алгоритмы электронной подписи, которые не позволяют восстанавливать зашифрованную ранее с их помощью информацию в процессе верификации. В частности, в составе OpenSSL реализован алгоритм из стандарта ГОСТ Р 34.10-2001, основанный на использовании эллиптических кривых.

Однако, ключевые пары такого алгоритма, принадлежащие двум участникам протокола защищенного обмена данными и операции над точками эллиптической кривой, определенные стандартом, могут быть использованы для реализации алгоритма согласования ключей VKO GOST R 34.10-2001, описанного в RFC 4357. Он представляет собой вариант алгоритма ECDH (Elliptic curve Diffie–Hellman, алгоритм Диффи-Хелмана на эллиптических кривых). Согласованный с его помощью 256-битный ключ далее используется для шифрования сеансового ключа с помощью блочного шифра ГОСТ 28147-89. Далее будет рассмотрен порядок генерации ключевых пар алгоритма ГОСТ Р 34.10-2001, их загрузки в файл и выгрузки из файла, а также непосредственно процесс согласования ключа шифрования сеансового ключа.

Согласование ключа с помощью алгоритма VKO GOST R 34.10-2001

Для реализации процедуры согласования ключа будем использовать функции, прототипы которых описаны в заголовочном файле openssl/evp.h. Сначала рассмотрим процесс генерации ключевых пар алгоритма электронной подписи ГОСТ Р 34.10-2001.

Первым делом необходимо создать и инициализировать контекст для операций с ключевыми парами. Для их хранения применяется структура EVP\_PKEY. В программе объявляется переменная-указатель на эту структуру, а сама она (пока пустая без каких-либо параметров и значений ключевой пары) создается в динамической памяти с помощью функции:

EVP\_PKEY \*EVP\_PKEY\_new(void);

Если в дальнейшем потребуется освободить память, выделенную под структуру типа EVP\_PKEY, то можно использовать функцию:

void EVP\_PKEY\_free(EVP\_PKEY \*pkey);

Также необходимо объявить указатель еще на одну структуру: EVP\_PKEY\_CTX. Она содержит непосредственно сам контекст для выполнения операций с ключевыми парами. Для создания контекста, который будет использоваться при генерации ключевой пары, необходимо использовать функцию:

EVP\_PKEY\_CTX \*EVP\_PKEY\_CTX\_new\_id(int id, ENGINE \*e);

Параметр e, представляет собой указатель на структуру ENGINE, также предварительно получаемый с помощью функции ENGINE\_by\_id. Параметр id является предопределенным числовым идентификатором (NID), который соответствует нужному алгоритму. Для алгоритма ГОСТ Р 34.10-2001 это NID\_id\_GostR3410\_2001. Функция возвращает значение, которое нужно присвоить указателю на структуру типа EVP\_PKEY\_CTX. В случае ошибки вернется NULL.

Перед генерацией ключевой пары необходимо установить набор параметров алгоритма. Наборы определены в RFC 4357 и обозначаются одной или двумя латинскими буквами: A, B, C, XA, XB (можно также указывать соответствующий OID). Какой из наборов использовать в данном случае без разницы, лишь бы они совпадали в обеих ключевых парах, так как эти параметры используются при вычислении общего ключа. Установить набор можно с помощью функции:

int EVP\_PKEY\_CTX\_ctrl\_str(EVP\_PKEY\_CTX \*ctx,

const char \*type, const char \*value);

Функция передает контексту, ранее полученному с помощью вызова функции EVP\_PKEY\_CTX\_new\_id, команду с названием, представляемым строковой константой (параметр type). Для алгоритма ГОСТ Р 34.10-2001 имеется только одна команда – "paramset". В качестве параметра value передается строковая константа с названием набора параметров, например "A".

После установки параметров алгоритма, необходимо инициализировать контекст для создания ключевой пары. Это делается с помощью вызова функции:

int EVP\_PKEY\_keygen\_init(EVP\_PKEY\_CTX \*ctx);

Генерация ключевой пары осуществляется вызовом функции:

int EVP\_PKEY\_keygen(EVP\_PKEY\_CTX \*ctx, EVP\_PKEY \*\*ppkey);

В качестве параметров функции передается указатель на структуру с контекстом и адрес указателя (так как параметр изменяется) на структуру для размещения сгенерированной ключевой пары.

После генерации ключевой пары ее необходимо сохранить в файлах, содержащих открытый и закрытый ключ. Также как и для хранения ключей алгоритма RSA, будем использовать выгрузку в файл PEM-формата. Для сохранения в файле закрытого ключа можно использовать функцию:

int PEM\_write\_bio\_PrivateKey(BIO \*bp, EVP\_PKEY \*x, const EVP\_CIPHER \*enc, unsigned char \*kstr, int klen, pem\_password\_cb \*cb, void \*u);

Параметр x является указателем на структуру типа EVP\_PKEY, содержащую сгенерированную ключевую пару. Остальные параметры аналогичны по своему назначению одноименным параметрам функции PEM\_write\_bio\_RSAPrivateKey, рассмотренной ранее.

Выгрузка в файл открытого ключа осуществляется вызовом функции:

int PEM\_write\_bio\_PUBKEY(BIO \*bp, EVP\_PKEY \*x);

Ее параметры аналогичны одноименным у рассмотренной выше функции PEM\_write\_bio\_PrivateKey.

Загрузка закрытого ключа из файла осуществляется функцией:

EVP\_PKEY \*PEM\_read\_bio\_PrivateKey(BIO \*bp, EVP\_PKEY \*\*x,

pem\_password\_cb \*cb, void \*u);

В качестве параметра x передается адрес указателя на структуру типа EVP\_PKEY, в которую будет записан извлекаемый из файла закрытый ключ. Назначение остальных параметров аналогично функции PEM\_read\_bio\_RSAPrivateKey. Возвращаемое функцией значение присваивается указателю на структуру типа EVP\_PKEY. В случае ошибки возвращается NULL.

Для загрузки открытого ключа используется функция:

EVP\_PKEY \*PEM\_read\_bio\_PUBKEY(BIO \*bp, EVP\_PKEY \*\*x,

pem\_password\_cb \*cb, void \*u);

Параметры функции аналогичны рассмотренной выше функции PEM\_read\_bio\_PrivateKey. Последние два параметра всегда задаются равными NULL. Функция возвращает указатель на структуру типа EVP\_PKEY, в которую будет записан извлекаемый из файла открытый ключ. В случае ошибки возвращается NULL.

После генерации ключевой пары и выгрузки ее в файлы контекст должен быть освобожден функцией:

void EVP\_PKEY\_CTX\_free(EVP\_PKEY\_CTX \*ctx);

Далее рассмотрим процесс выработки сторонами обмена информацией общего ключа, который можно будет использовать для зашифрования псевдослучайного сеансового ключа симметричным криптоалгоритмом ГОСТ 28147-89. Для этого каждой из сторон требуются следующие данные:

1. Собственный закрытый ключ.

2. Открытый ключ другой стороны.

3. 64-битная псевдослучайная величина UKM (user key material), которая генерируется одной из сторон и передается другой стороне в открытом виде.

Алгоритм может использовать не только статические ключевые пары, но и эфемерные (временные). Мы будем далее полагать, что используются статические (постоянные) ключевые пары. Для выработки общего ключа каждая сторона создает контекст на основе своего закрытого ключа с помощью функции:

EVP\_PKEY\_CTX \*EVP\_PKEY\_CTX\_new(EVP\_PKEY \*pkey, ENGINE \*e);

Параметр pkey указывает на структуру, содержащую закрытый ключ ключевой пары. Если он хранится в файле, то его необходимо в эту структуру предварительно загрузить. Функция возвращает указатель на созданный контекст или NULL в случае ошибки.

Далее созданный контекст необходимо инициализировать для выработки общего ключа с помощью функции:

int EVP\_PKEY\_derive\_init(EVP\_PKEY\_CTX \*ctx);

Теперь необходимо сгенерировать и установить в качестве параметра алгоритма величину UKM. Первоначальную генерацию данной величины осуществляет та сторона, которая генерирует сеансовый ключ, и, следовательно, первой начинает процесс согласования ключа. Произведя установку параметра в своем контексте, она передает его в дальнейшем другой стороне в открытом виде вместе с зашифрованным сеансовым ключом и зашифрованными им данными.

Для установки параметра UKM можно использовать функцию:

int EVP\_PKEY\_CTX\_ctrl(EVP\_PKEY\_CTX \*ctx, int keytype,

int optype, int cmd, int p1, void \*p2);

Функция позволяет задать команду для реализации алгоритма, на основе ключа которого создан контекст. В качестве параметров keytype (тип ключа) и optype (тип операции) будем указывать значение -1. В качестве параметра cmd (код команды) будем указывать предопределенную константу EVP\_PKEY\_CTRL\_SET\_IV. В качестве параметров команды используются величины p1 и p2. Первая, в данном случае, задает размерность буфера с данными для команды в байтах, а вторая является собственно указателем на этот буфер. Ниже показан пример установки параметра UKM:

unsigned int64 ukm; RAND\_bytes((unsigned char \*)&ukm, 8);

ret = EVP\_PKEY\_CTX\_ctrl(ctx, -1, -1, EVP\_PKEY\_CTRL\_SET\_IV,

8, &ukm);

После установки параметра UKM необходимо добавить к контексту выработки общего ключа открытый ключ другой стороны. Это делается с помощью функции:

int EVP\_PKEY\_derive\_set\_peer(EVP\_PKEY\_CTX \*ctx,

EVP\_PKEY \*peer);

Параметр peer является указателем на структуру, содержащую загруженный из файла открытый ключ другой стороны.

Далее можно непосредственно выработать общий ключ с помощью функции:

int EVP\_PKEY\_derive(EVP\_PKEY\_CTX \*ctx, unsigned char \*key,

size\_t \*keylen);

Параметр key является указателем на буфер размера keylen, в который помещается сгенерированный ключ. Если параметр key задать равным NULL, то по адресу keylen будет передан требуемый размер буфера. Для данного алгоритма генерируется 256-битный ключ, на основе которого создается контекст алгоритма ГОСТ 28147-89 и производится шифрование сеансового ключа. Зашифрованный сеансовый ключ выгружается в файл и отправляется другой стороне. Та, в свою очередь, получив величину UKM и зашифрованный сеансовый ключ, может сгенерировать общий ключ, расшифровать сеансовый ключ и использовать его для расшифрования присланных данных.

СОДЕРЖАНИЕ РАБОТЫ

Задание А

Разработать на языке программирования C/C++ с использованием средств криптографического пакета OpenSSL консольное или оконное приложение, выполняющее следующие функции:

1. Зашифрование/расшифрование указанного файла блочным шифром AES со 128-разрядным ключом в режиме CBC или CFB-128 на выбор пользователя. Сеансовый ключ и вектор инициализации генерируются псевдослучайно (ключ может также импортироваться, см. п. 4). Вектор должен сохраняться с файлом шифртекста (формат придумать самостоятельно).

2. Генерация ключевой пары RSA с длиной ключа не менее 2048 бит.

3. Сохранение открытого и закрытого ключей в файлах PEM-формата.

4. Зашифрование сеансового ключа шифра AES с помощью ранее сгенерированного открытого ключа RSA и сохранение его в файле, извлечение из файла и расшифрование с помощью закрытого ключа для восстановления ранее зашифрованного им файла.

Задание Б

Разработать на языке программирования C/C++ с использованием средств криптографического пакета OpenSSL консольное или оконное приложение, выполняющее следующие функции:

1. Зашифрование/расшифрование указанного файла блочным шифром ГОСТ 28147-89 в режиме CFB (устанавливается по умолчанию). Сеансовый ключ и вектор инициализации генерируются псевдослучайно (ключ может также импортироваться, см. п. 5). Вектор должен сохраняться с файлом шифртекста (формат придумать самостоятельно) и использоваться при расшифровании.

2. Генерация двух ключевых пар алгоритма ГОСТ Р 34.10-2001 с заранее определенным набором параметров.

3. Сохранение открытого и закрытого ключей в файлах PEM-формата.

4. Выработку общего для двух ключевых пар ключа симметричного шифрования, используемого для обмена сеансовым ключом. При этом величина UKM должна вырабатываться псевдослучайно, сохраняться в файле, а затем считываться при генерации общего ключа «второй стороной».

5. Зашифрование/расшифрование сеансового ключа блочным шифром ГОСТ 28147-89 на базе общего ключа, выработанного в п. 4, сохранение его в файле, извлечение из файла. Для зашифрования сеансового ключа можно использовать тот же вектор инициализации, что и для шифрования файла с открытым текстом.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего используется криптографический пакет OpenSSL?

2. В каких операционных системах можно использовать пакет OpenSSL?

3. Как установить и сконфигурировать пакет OpenSSL.

4. Как выполняется инициализация библиотеки шифрования?

5. Какие статические и динамические библиотеки пакета OpenSSL задействуются в данной работе?

6. Данные каких заголовочных файлов пакета OpenSSL используются в этой лабораторной работе?

7. Что собой представляет тип BIO? Какие его разновидности вы знаете?

8. Какими средствами в пакете OpenSSL можно осуществлять генерацию псевдослучайных чисел?

9. Какие функции и типы данных, необходимые для выполнения симметричного шифрования алгоритмом AES, вы знаете?

10. Как можно осуществлять асимметричное шифрование алгоритмом RSA средствами пакета OpenSSL?

11. Какие функции для файловой выгрузки-загрузки открытых и закрытых ключей ключевых пар алгоритма RSA вы знаете?

12. Как активировать поддержку отечественных криптоалгоритмов в пакете OpenSSL?

13. Какие функции и типы данных используются при шифровании криптоалгоритмом ГОСТ 28147-89?

14. Как в отечественной криптографии строится процесс обмена сеансовым ключом?

15. Как осуществляется генерация ключевых пар алгоритма электронной подписи ГОСТ Р 34.10-2001?

16. Какие функции используются для загрузки в файл и выгрузки из файла ключевых пар алгоритма электронной подписи ГОСТ Р 34.10-2001?

17. Какие параметры используются при выработке общего ключа с помощью алгоритма VKO GOST R 34.10-2001? Какие функции и типы данных используются для реализации этого алгоритма?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 11**

**СОЗДАНИЕ КРИПТОГРАФИЧЕСКИХ СООБЩЕНИЙ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА MICROSOFT CRYPTOAPI И ЦИФРОВЫХ СЕРТИФИКАТОВ X.509**

Цель работы: ознакомиться со структурой и форматами представления сертификатов открытых ключей, способами их создания и импортирования в систему, а также получить навыки в создании криптографических сообщений средствами интерфейса Microsoft CryptoAPI.

ОСНОВНЫЕ ПОНЯТИЯ

Общие сведения о сертификатах X.509

Генерация ключей с использованием случайных (псевдослучайных) чисел является стандартным подходом при использовании средств симметричной криптографии. А обмен сеансовым ключом между пользователями обычно производится средствами асимметричной криптографии. Один из пользователей может, получив свободно распространяемый открытый ключ другого пользователя, зашифровать им сгенерированный сеансовый ключ, создав так называемый

«цифровой конверт». Далее зашифрованный сеансовый ключ отправляется владельцу ключевой пары и тот с помощью закрытого ключа расшифровывает его и использует в дальнейшем обмене сообщениями.

Главная проблема такого подхода состоит в том, что не обеспечивается аутентификация пользователей. То есть пользователь не может быть уверен, что присланный ему открытый ключ действительно принадлежит заявленному лицу. На этом построена атака типа «человек посередине», когда злоумышленник перехватывает пересылаемый открытый ключ легального пользователя и заменяет его своим. Это позволяет ему в дальнейшем, перехватывая пересылаемые сообщения и перезашифровывая их ключом одного из легальных пользователей, получать доступ ко всей переписке.

Для устранения проблемы взаимной аутентификации пользователей и был разработан стандарт Международного союза телекоммуникаций (ITU) X.509. Он включает в себя описание элементов так называемых инфраструктур открытых ключей (Public

Key Infrastructure, PKI), а также процедур распределения ключей. Основным элементом схемы аутентификации являются сертификаты открытых ключей, содержащие сведения о владельце ключевой пары и его открытый ключ.

Сертификаты выдаются пользователям центрами сертификации (ЦС, Certification Authority – CA), сведения о которых также имеются в составе сертификата. ЦС подписывает сертификат пользователя своим закрытым ключом и далее любой желающий может проверить подлинность сертификата, верифицируя электронную подпись (ЭП) центра с помощью его открытого ключа. Таким образом, пользователю для проверки подлинности сертификата другого пользователя, нужен сертификат его ЦС. И в этом случае основным становится вопрос доверия пользователя к сертификату самого ЦС.

Как правило, отдельная PKI может развертываться в рамках корпоративной сети предприятия и содержать несколько ЦС, которые чаще всего связаны в иерархическую структуру. На вершине иерархии

– корневой ЦС, имеющий, как правило, самоподписанный сертификат, то есть содержащий в своем составе ЭП, созданную с помощью своего же закрытого ключа. Этот ЦС выдает сертификаты подчиненным ЦС, которые в свою очередь уже могут выдавать сертификаты конечным пользователям или сертифицировать нижестоящие центры. В общем случае, если пользователи сертифицированы разными ЦС, то процесс проверки подлинности сертификатов может привести к проверке цепочки сертификатов ЦС, образующих путь в данной иерархии к узлу, которому пользователь доверяет. Такой узел будет называться доверенным корневым центром и доверие к нему будет означать доверие и ко всем нижестоящим в этой иерархии ЦС.

В настоящее время применяется третья версия стандарта (X.509 v3), согласно которой в состав сертификата входит ряд полей данных:

 Version (номер версии) – указывается десятичное значение 3 и шестнадцатеричное 0x2.

 Serial Number (серийный номер) – целочисленное значение, уникальное для данного ЦС.

 Signature Algorithm (идентификатор алгоритма подписи) – поле, определяющее использованные ЦС при создании сертификата алгоритмы хэширования и цифровой подписи.

 Issuer (эмитент, издатель) – поле, содержащее отличительное имя (Distinguished Name, DN) центра сертификации, выдавшего сертификат. Формат записи отличительных имен определен стандартом X.500. Он состоит из набора выражений типа

«атрибут=значение», разделенных запятой. Например,

отличительное имя ЦС может выглядеть так: C=RU, ST=Belgorodskaya obl., L=Belgorod, O=BSTU, CN=IT\_CA. В данной записи использованы атрибуты: C (Country Name) – двухбуквенный код страны, ST (State or Province Name) – наименование области, L (Locality Name) – наименование населенного пункта, O (Organization Name) – название организации, CN (Common Name) – общепринятое имя.

 Validity (Not Before/ Not After) (период действия (не ранее/не позднее)) – значения, определяющие период, на протяжении которого сертификат действителен.

 Subject (субъект) – отличительное имя субъекта (владельца ключевой пары). У самоподписанных сертификатов значения полей Issuer и Subject совпадает.

 Subject Public Key Info (информация об открытом ключе субъекта) – содержит значение открытого ключа и идентификатор алгоритма.

 Расширения (необязательные поля, определенные в версии 3) – могут содержать информацию, которую можно разделить на три категории: информация о ключах и политиках, атрибуты субъекта и органа сертификации, ограничения маршрута сертификации. Поля могут объявляться критичными и некритичными. Некритичные поля приложение, использующее сертификат, может игнорировать.

 Значение подписи сертификата – представляет собой подписанный закрытым ключом ЦС хэш-код всех полей сертификата.

Для хранения сертификатов в запоминающих устройствах и оперирования ими приложениями разных типов необходимы единообразные форматы их представления и кодирования. Для этого применяется так называемая абстрактная синтаксическая нотация версии 1 (Abstract Syntax Notation One, ASN.1). ASN.1 является гибкой нотацией, позволяющей определять как простые, так и структурированные типы данных и кодировать их совокупностью байтов (октетов). Для представления содержимого сертификатов в рамках нотации ASN.1 используются отличительные правила кодирования (Distinguished Encoding Rules, DER), которые обеспечивают однозначный способ кодирования каждого из значений ASN.1.

Что касается форматов файлов, содержащих сертификаты, то здесь имеется несколько вариантов. Некоторые из этих форматов представления сертификатов определены в стандарте PEM (Privacy

Enhanced Mail, почта повышенной секретности) и группе стандартов PKCS (Public Key Cryptography Standards, стандарты криптографии с открытым ключом компании RSA Security, Inc.). Перечень форматов представлен ниже:

 DER подразумевает хранение в файле непосредственно двоичного содержимого сертификата в DER-кодировке. Файлы обычно имеют расширение .cer или .crt и могут содержать только один сертификат без пути сертификации (цепочки сертификатов, ведущей к доверенному корневому ЦС).

 PEM – это сертификат в формате DER, закодированный с помощью кодировки base64, которая позволяет представить произвольные двоичные данные в виде последовательности печатных ASCII-символов. Закодированные данные помещаются в файле между строками « BEGIN

CERTIFICATE-----» и

«-----END CERTIFICATE-----», которые используются как ограничители начала и конца сертификата. По умолчанию файлы имеют расширение .pem, но можно также использовать и расширения .cer и .crt. Данный формат также не позволяет задавать доверительную цепочку сертификатов. Кроме сертификатов, в этом формате хранятся закрытые ключи (предварительно зашифрованные симметричным алгоритмом с ключом из хэшированного пароля) и запросы на сертификацию (см. ниже). Назначение содержимого файла можно узнать из названий ограничителей.

 PKCS#7 изначально предназначался для определения синтаксиса криптографических сообщений (Cryptographic Message Syntax, CMS), в которых применялись бы шифрование и/или ЭП. Однако можно создать и вырожденное сообщение, которое не имеет данных, а содержит только сами сертификаты. Кроме того, в файле этого формата можно разместить все сертификаты, входящие в доверенный путь сертификации. Файлы имеют расширение .p7b или .p7c.

 PKCS#12 является единственным вариантом хранения сертификата, закрытого ключа и доверительной цепочки сертификации в одном файле. Он используется в основном для экспорта закрытого ключа. Данные шифруются симметричным алгоритмом с ключом из хэшированного пароля. Файл может иметь расширение .p12 или .pfx.

Кроме вышеперечисленного, также для создания запроса пользователя на сертификацию применяется формат PKCS#10. В

запросе указываются данные субъекта, его открытый ключ, необходимые параметры и далее запрос подписывается закрытым ключом субъекта. Хранится запрос обычно в файле PEM-формата с соответствующими названиями ограничителей.

Стандарт X.509 помимо сертификатов определяет много и других элементов инфраструктур открытых ключей. Однако в данной работе мы не будем рассматривать работу реальной PKI, а ограничимся использованием сертификатов по сути, как контейнеров ключей, которые могут устанавливаться в операционной системе и использоваться для защищенного обмена данными. Будем считать, что созданные в процессе выполнения работы сертификаты принадлежат простейшей PKI, содержащей один ЦС с самоподписанным сертификатом.

В составе настольных версий ОС Windows имеются средства для управления сертификатами, но нет средств их создания. В данной работе для этих целей мы будем использовать криптографический пакет OpenSSL.

Создание сертификатов X.509 с помощью OpenSSL

Пакет OpenSSL позволяет развертывать PKI, что подразумевает возможность создания сертификатов X.509 и дальнейшего оперирования ими. Воспользуемся этим функционалом пакета, осуществляя взаимодействие с ним через командную строку в режиме консольного приложения. Далее вкратце опишем процесс создания сертификатов X.509.

В прошлой работе была показана процедура настройки пакета с помощью конфигурационного файла. Внесем дополнительные изменения в ряд секций этого файла с помощью любого текстового редактора. В частности, в секцию [ req\_distinguished\_name ], которая содержит значения переменных, определяющих параметры отличительных имен, указываемых в запросах на сертификацию. Установим новые значения переменных, отвечающих за значения атрибутов имен по умолчанию:

countryName\_default = RU stateOrProvinceName\_default = Belgorodskaya obl. localityName\_default = Belgorod 0.organizationName\_default = My\_organization

Содержимое секции [ req\_attributes ] удалим полностью (но название секции оставим!). Секция [ policy\_match ] определяет какие из элементов отличительного имени субъекта в запросе на

сертификацию должны совпадать с соответствующими элементами имени ЦС. По умолчанию совпадать должны названия страны, области и организации (параметры установлены в значение match). При желании их можно изменить на значения supplied (предоставленный) и optional (необязательный).

OpenSSL предоставляет возможность использования тестового ЦС для создания сертификатов открытых ключей. В конфигурационном файле в секции [ CA\_default ] определены параметры по умолчанию работы пакета в режиме ЦС. В частности параметр dir, который определяет местоположение каталога ЦС, изначально установлен в значение ./demoCA. Вместо этого введем непосредственный путь к каталогу, например:

dir = C:/OpenSSL-Win32/demoCA

После этого сохраним конфигурационный файл и закроем его. Теперь по указанному пути создадим каталог с именем demoCA.

Далее необходимо создать набор файлов и подкаталогов, требуемый для функционирования тестового ЦС. Сначала в созданном каталоге demoCA создадим два текстовых файла: пустой с именем index.txt и файл serial со значением серийного номера (например, 01), который будет присвоен следующему подписанному сертификату. Также в каталоге demoCA создадим подкаталог newcerts, в который будут помещаться копии создаваемых сертификатов.

В состав OpenSSL входит большое число команд, имеющих многочисленные параметры и их подробное рассмотрение в рамках данной работы невозможно. Поэтому рассмотрим несколько необходимых команд, которые понадобятся для выполнения данной работы.

Для выполнения работы нам понадобится создать:

 ключевую пару алгоритма RSA и самоподписанный сертификат ЦС в PEM-формате, которые будут использоваться для создания сертификатов конечных пользователей;

 две ключевые пары RSA и сертификаты пользователей в PEM- формате, подписанные созданным ранее закрытым ключом ЦС;

 сертификаты пользователей в формате PKCS#12, созданные на базе ключевых пар и сертификатов из предыдущего пункта. Они понадобятся непосредственно для создания и расшифрования криптографических сообщений;

Рассмотрим процесс создания данных сертификатов командами OpenSSL на конкретных примерах, с учетом изменений, внесенных ранее в конфигурационный файл. Все приведенные далее примеры

команд предполагают, что файл openssl.exe уже запущен и команды вводятся после приглашения OpenSSL>.

Создание самоподписанного сертификата выполним с помощью команды req. Вообще эта команда предназначена для создания запросов на сертификацию, но если указана опция -x509, то создается самоподписанный сертификат. Ниже показан пример команды, которая одновременно выполняет генерацию ключевой пары RSA с длиной 2048 бит (опция -newkey), задает срок действия сертификата в 2 года (опция -days), записывает созданную ключевую пару в файл ca\_test\_key.pem текущего каталога (опция -keyout), а сертификат – в файл ca\_test\_cert.pem (опция -out):

req -x509 -newkey rsa:2048 -days 730 -keyout ca\_test\_key.pem -out ca\_test\_cert.pem

После запуска команды необходимо ввести ряд данных. Сначала это ввод и подтверждение пароля, который будет использован для шифрования закрытого ключа. Его считывание происходит из стандартного потока ввода без эхо-вывода на экран и завершается после нажатия клавиши Enter. Затем пользователь должен ввести атрибуты отличительного имени:

Country Name (2 letter code) [RU]:

State or Province Name (full name) [Belgorodskaya obl.]: Locality Name (eg, city) [Belgorod]:

Organization Name (eg, company) [My\_organization]: Organizational Unit Name (eg, section) []:

Common Name (e.g. server FQDN or YOUR name) []:CA\_Test Email Address []:

Значения атрибутов вводятся латинскими буквами, ввод завершается нажатием клавиши Enter. Слева в квадратных скобках указываются значения по умолчанию, взятые из конфигурационного файла. Не вводя каких-либо символов и нажав клавишу Enter, пользователь соглашается со значением по умолчанию. Если значение по умолчанию в файле не указано, то скобки остаются пустыми. Нажав клавишу Enter в этом случае, пользователь игнорирует такой атрибут. В приведенном примере мы согласились со значениями по умолчанию для атрибутов: Country Name, State or Province Name, Locality Name, Organization Name, проигнорировали атрибуты: Organizational Unit Name, Email Address и установили для атрибута Common Name значение CA\_Test.

Созданные сертификат и закрытый ключ в файлах представлены только в кодировке base64. Увидеть сертификат в текстовом виде в окне консоли можно, введя команду:

x509 –in ca\_test\_cert.pem –noout –text

Выведя сертификат в окно консоли можно убедиться, что отличительные имена эмитента и субъекта совпадают. В разделе расширений можно увидеть, что в подразделе X509v3 Basic Constraints (основные ограничения) значение параметра CA установлено в значение TRUE, что означает принадлежность данного сертификата центру сертификации. У сертификата обычного пользователя этот параметр будет установлен в FALSE.

Для просмотра файла закрытого ключа используется аналогичная команда, но вместо x509 нужно указать rsa. После ввода команды необходимо будет указать парольную фразу.

Создание ключевой пары и сертификата пользователя проводится в два этапа. Сначала командой req создается запрос на сертификацию с одновременной генерацией ключевой пары, а затем с помощью команды ca он подписывается закрытым ключом ЦС. Создание запроса:

req -newkey rsa:2048 -keyout User\_A\_key.pem -out User\_A\_req.pem

Команда создает новый закрытый ключ RSA длиной 2048 бит, помещает его в файл User\_A\_key.pem, а запрос помещает в файл User\_A\_req.pem. Далее также вводится пароль и отличительное имя. Значения всех атрибутов (кроме Common Name) зададим аналогичными тем, которые содержатся в сертификате ЦС. В качестве значения атрибута Common Name укажем User\_A.

Теперь необходимо подписать созданный запрос на сертификацию с помощью сертификата ЦС и его закрытого ключа. Для этого будет использоваться команда ca, которая имитирует работу ЦС. Создание сертификата из ранее созданного запроса произведем командой:

ca -md sha256 -keyfile ca\_test\_key.pem -cert ca\_test\_cert.pem -in User\_A\_req.pem -out User\_A\_cert.pem

Данная команда определила использование для создания ЭП хэш- функции SHA-256, в качестве закрытого ключа ЦС указан файл ca\_test\_key.pem, в качестве сертификата ЦС файл ca\_test\_cert.pem, запрос на сертификацию взят из файла User\_A\_req.pem, а созданный сертификат помещается в файл User\_A\_cert.pem. После запуска команды необходимо ввести пароль для доступа к закрытому ключу

ЦС, а затем подтвердить создание ЭП и формирование сертификата на базе запроса. В выходном файле размещается сначала текстовая версия сертификата, а затем его вариант в формате PEM. Если бы был необходим сертификат без текстового варианта, то в предыдущую команду надо было добавить опцию -notext. Сертификат по умолчанию создается с периодом действия 1 год (параметр default\_days секции [ CA\_default ] конфигурационного файла изначально установлен в значение 365).

Теперь на базе сертификата пользователя в формате PEM и его закрытого ключа создадим сертификат в формате PKCS#12:

pkcs12 -export -in User\_A\_cert.pem -inkey User\_A\_key.pem

-out User\_A\_cert.p12

После запуска команды необходимо будет ввести пароль доступа к закрытому ключу в файле User\_A\_key.pem и новый пароль экспорта для дальнейшего доступа к созданному сертификату.

Аналогично создадим ключевую пару, сертификаты в форматах PEM и PKCS#12 для второго пользователя (User\_B).

Созданные сертификаты далее будут использованы для создания и расшифрования криптографических сообщений. Но сначала их необходимо импортировать в состав ОС Windows.

Управление сертификатами в ОС Windows

Для безопасного хранения сертификатов Windows использует хранилище сертификатов отдельно для каждого пользователя, так чтобы другой пользователь не смог ими воспользоваться. Физическое хранилище разделено на несколько логических. Каждое такое хранилище предназначено для сертификатов определенных функций и назначений.

Для того чтобы импортировать сертификат ЦС (ca\_test\_cert.pem), изменим его расширение с .pem на .crt. После в окне Проводника дважды щелкнем по значку этого файла. В появившемся диалоговом окне отображены сведения о сертификате, а также сообщение о том, что к этому корневому сертификату нет доверия. Если нажать на кнопку Установить сертификат, то появится окно Мастера импорта сертификатов. В этом окне нажмем кнопку Далее, после чего появится окно следующего шага, где нужно определить в какое логическое хранилище будет помещен импортируемый сертификат. Установим зависимый переключатель в положение Поместить все сертификаты в следующее хранилище и нажмем кнопку Обзор. В

появившемся списке выберем Доверенные корневые центры сертификации и нажмем OK, затем Далее. В завершающем окне нажмем кнопку Готово. На экране появится окно с предупреждением о безопасности, в котором сказано, что не удается проверить данный сертификат и необходимо выбрать, устанавливать ли данный сертификат или нет. Нажав Да завершим процесс установки.

Теперь установим сертификаты в формате PKCS#12. Сначала дважды щелкнем по файлу User\_A\_cert.p12. В первом окне Мастера нажмем кнопку Далее. В следующем окне, где предлагается уточнить имя импортируемого файла, также нажмем Далее. В следующем окне необходимо ввести пароль, который был указан при создании сертификата. Также установим флажок Пометить этот ключ как экспортируемый, так как в дальнейшем нам может понадобится экспортировать ключевую пару. Далее выберем в качестве хранилища Личное и завершим процесс импорта. Аналогично установим сертификат из файла User\_B\_cert.p12.

Для управления установленными сертификатами в Windows имеется оснастка Сертификаты. Отобразить ее можно запустив на выполнение файл certmgr.msc. В ее окне слева расположен перечень логических хранилищ. Для того чтобы просмотреть содержимое хранилища Личное раскроем его структуру и щелкнем по значку Сертификаты. Справа будут отображены установленные сертификаты с именами User\_A и User\_B, которые мы только что установили. Можно увидеть имя выдавшего их ЦС и срок действия. Используя меню Действия можно открыть сертификат, просмотреть его свойства, копировать, удалить и т.д. Если раскрыть хранилище Доверенные корневые центры сертификации, то можно просматривая его содержимое увидеть и установленный нами сертификат с именем CA\_Test.

Создание криптографических сообщений с помощью функций интерфейса CryptoAPI 2.0

В предыдущей работе процесс защищенного обмена сообщениями предполагал создание отдельных файлов с зашифрованным сообщением, зашифрованным сеансовым ключом и открытым ключом, которые могли пересылаться между двумя пользователями. При этом форматы пересылаемых данных должны были заранее согласовываться и для разных групп пользователей они могли быть различными.

Для стандартизации таких форматов RSA Laboratories предложила спецификацию PKCS#7 (RFC 2315), приемником которой стал выпущенный IETF стандарт «Cryptographic Message Syntax (CMS)» (RFC 5652). Стандарт CMS определяет структуру криптографических сообщений, которые могут содержать в себе зашифрованные и/или подписанные данные вместе со всей необходимой для дальнейшего расшифрования и/или проверки ЭП информацией. Например, при создании зашифрованного сообщения, в его состав включается зашифрованный открытым ключом получателя сообщения сеансовый ключ. В качестве источников ключевой информации асимметричных алгоритмов при создании криптографических сообщений используются установленные в операционной системе сертификаты. Мы будем использовать ранее сформированные и установленные в системе сертификаты пользователей User\_A и User\_B, а для проверки их действительности сертификат центра сертификации CA\_Test.

В данной работе необходимо реализовать приложение, создающее и расшифровывающее криптографическое сообщение, содержащее электронную подпись сообщения и результат его зашифрования симметричным алгоритмом. Процесс создания такого сообщения можно представить следующим обобщенным алгоритмом:

1. Открыть системные хранилища сертификатов «Личное» (MY) и

«Доверенные корневые центры сертификации» (ROOT). В первое мы установили сертификаты пользователей, которые будут участвовать в «обмене» сообщениями, а второе содержит сертификат ЦС. Для простоты в нашем случае сертификат получателя сообщения содержит его полную ключевую пару, хотя в реальной ситуации, конечно же, отправитель будет располагать только сертификатом с открытым ключом получателя.

2. Сформировать и предоставить пользователю список имен владельцев сертификатов, установленных в хранилище

«Личное».

3. Получить выбранные пользователем из списка имена отправителя и получателя криптографического сообщения.

4. Получить контексты сертификатов отправителя и получателя сообщения и проверить их целостность с помощью сертификата ЦС.

5. Считать из файла исходное сообщение и поместить его в буфер, созданный в динамической памяти.

6. Инициализировать параметры функции, используемой для создания криптографического сообщения и получить объем

буфера в динамической памяти, необходимого для хранения блоба с сообщением.

7. Создать в динамической памяти буфер требуемого размера и вызвать функцию создания криптографического сообщения, используя в качестве одного из параметров указатель на созданный выходной буфер.

8. Сохранить созданный блоб в указанном пользователем файле.

Для расшифрования криптографического сообщения нужно выполнить следующие действия:

1. Открыть системное хранилище, которое содержит сертификат получателя (с ключевой парой) и сертификат отправителя (с открытым ключом). В нашем случае это хранилище «Личное». Сертификат получателя нужен для того, чтобы расшифровать с помощью закрытого ключа зашифрованный сеансовый ключ, а сертификат отправителя с его открытым ключом, чтобы проверить ЭП из состава криптографического сообщения. Для простоты мы не устанавливали в системе отдельный сертификат отправителя с открытым ключом, поэтому при расшифровании будет задействован тот же сертификат в формате PKCS#12, который участвовал в создании сообщения. В состав сообщения помещаются данные о владельцах сертификатов, которые при расшифровании ищутся в указанном хранилище.

2. Считать из файла блоб с криптографическим сообщением и поместить его в буфер, созданный в динамической памяти.

3. Инициализировать параметры функции, используемой для расшифрования криптографического сообщения и получить объем буфера в динамической памяти, необходимого для хранения блоба с расшифрованным сообщением.

4. Создать в динамической памяти буфер требуемого размера и вызвать функцию расшифрования криптографического сообщения, используя в качестве одного из параметров указатель на созданный выходной буфер.

5. Проверить результат расшифрования и верификации ЭП. В случае успеха сохранить расшифрованное сообщение в указанном пользователем файле.

Детали этих алгоритмов будут поясняться по мере рассмотрения функций интерфейса CryptoAPI, которые используются для их реализации. Большинство из них относится к CryptoAPI версии 2.0, поэтому в создаваемом приложении необходимо подключить библиотеку crypt32.dll. В среде Visual Studio для этого можно

использовать библиотеку импорта crypt32.lib, указав этот файл в свойствах проекта или включив в текст программы директиву:

#pragma comment(lib, "crypt32.lib")

Что касается заголовочных файлов, то в текст программы необходимо включить файлы windows.h и wincrypt.h.

Все рассмотренные далее функции можно разделить на три типа:

1. Функции управления хранилищами сертификатов:

CertOpenSystemStore, CertCloseStore.

2. Функции для работы с сертификатами: CertEnumCertificatesInStore, CertGetNameString, CertFindCertificateInStore, CertGetIssuerCertificateFromStore, CertFreeCertificateContext.

3. Функции поддержки криптографических сообщений. В CryptoAPI существуют два вида таких функций: базовые и упрощенные. Необходимость использования базовых возникает достаточно редко, поэтому мы рассмотрим только упрощенные функции: CryptSignAndEncryptMessage, CryptDecryptAndVerifyMessageSignature.

Более подробная информация о рассмотренных функциях содержится в разделе MSDN, посвященном использованию CryptoAPI (https://msdn.microsoft.com/en-us/library/windows/desktop/aa380256%28 v= vs.85%29.aspx).

CertOpenSystemStore

Функция открывает системное хранилище сертификатов и имеет следующий прототип:

HCERTSTORE WINAPI CertOpenSystemStore(

HCRYPTPROV\_LEGACY hprov,

LPTCSTR szSubsystemProtocol );

Параметр hprov не используется и должен быть равен NULL. Строка szSubsystemProtocol содержит имя системного хранилища сертификатов. В данной работе в качестве имени будем использовать строку “MY” для открытия хранилища «Личное» и строку “ROOT” для открытия хранилища «Доверенные корневые центры сертификации».

В случае успешного завершения функция возвращает дескриптор открытого хранилища. В противном случае функция возвращает NULL. Открытое хранилище должно быть позднее закрыто функцией CertCloseStore (см. ниже). Пример вызова функции для открытия хранилища «Личное»:

HCERTSTORE hStoreMy = NULL;

if (!( hStoreMy = CertOpenSystemStore(

NULL,TEXT("MY"))))

{

//Вывод сообщения об ошибке

}

CertCloseStore

Функция закрывает открытое ранее хранилище сертификатов и имеет следующий прототип:

BOOL WINAPI CertCloseStore(

HCERTSTORE hCertStore, DWORD dwFlags );

Параметр hCertStore задает дескриптор закрываемого хранилища. Параметр dwFlags будем задавать равным нулю. Функция возвращает TRUE в случае успеха и FALSE в случае неудачи.

CertEnumCertificatesInStore

Функция используется для перечисления всех контекстов сертификатов, которые хранятся в указанном хранилище. Имеет прототип:

PCCERT\_CONTEXT WINAPI CertEnumCertificatesInStore(

HCERTSTORE hCertStore, PCCERT\_CONTEXT pPrevCertContext );

Параметр hCertStore задает дескриптор нужного хранилища, а pPrevCertContext – это указатель на структуру типа CERT\_CONTEXT, которая содержит контекст предыдущего сертификата, найденного в данном хранилище. Функция возвращает указатель на контекст очередного сертификата, извлеченного из хранилища или NULL, после извлечения всех сертификатов (или если хранилище изначально пустое). Ниже показан пример организации перечисления сертификатов в

PCCERT\_CONTEXT pCert = NULL;

while (pCert = CertEnumCertificatesInStore(hStoreMy, pCert))

{

//Обработка извлеченного контекста сертификата

}

CertGetNameString

Функция извлекает из переданного контекста сертификата и конвертирует в строку с нулем в конце имя владельца или издателя (ЦС). В зависимости от заданных параметров, функция в качестве результата своей работы может возвращать различные части отличительного имени или какой-либо вариант альтернативного имени, заданного в соответствующем поле расширения 3-й версии стандарта X.509. Данную функцию можно использовать, например, для получения имени субъекта сертификата, извлеченного из хранилища предыдущей функцией. Имеет прототип:

DWORD WINAPI CertGetNameString(

PCCERT\_CONTEXT pCertContext, DWORD dwType,

DWORD dwFlags, void \*pvTypePara,

LPTSTR pszNameString, DWORD cchNameString );

Параметр pCertContext – это указатель на структуру типа CERT\_CONTEXT, которая содержит контекст обрабатываемого сертификата. Параметр dwType определяет, какую часть отличительного имени (или альтернативного) будет возвращать функция. Для получения общепринятого имени (CN) будем использовать в качестве этого параметра константу CERT\_NAME\_SIMPLE\_DISPLAY\_TYPE. Для получения имени субъекта параметр dwFlags будем задавать равным нулю. Также равным NULL будем задавать параметр pvTypePara. Параметр pszNameString является адресом буфера, принимающего возвращаемую функцией строку. Количество символов, которое может вместить данный буфер передается через параметр cchNameString.

Функция возвращает количество символов, переданных в буфер pszNameString, исключая нуль-символ. Если требуемая часть имени в сертификате отсутствует, то функция вернет единицу и по адресу pszNameString будет пустая строка. Пример вызова функции:

TCHAR szNameString[128]; if (CertGetNameString(

pCert, CERT\_NAME\_SIMPLE\_DISPLAY\_TYPE, 0, NULL,

szNameString, 128) > 1)

{

//Обработка строки szNameString

}

CertFindCertificateInStore

Функция ищет в указанном хранилище первый (или следующий) сертификат, параметры которого совпадают с заданными критериями поиска. Например, эту функцию можно использовать, если нужно найти и извлечь из хранилища сертификат (или несколько сертификатов) по заданному общепринятому имени субъекта (CN). Прототип функции имеет вид:

PCCERT\_CONTEXT WINAPI CertFindCertificateInStore(

HCERTSTORE hCertStore, DWORD dwCertEncodingType, DWORD dwFindFlags,

DWORD dwFindType,

const void \*pvFindPara, PCCERT\_CONTEXT pPrevCertContext );

Параметр hCertStore является дескриптором открытого хранилища сертификатов. В качестве параметра dwCertEncodingType указывается результат побитового ИЛИ между константами X509\_ASN\_ENCODING и PKCS\_7\_ASN\_ENCODING. Параметр dwFindFlags задается равным нулю. Параметр dwFindType может содержать одну из предопределенных констант, определяющих тип поиска. Мы будем задавать этот параметр равным константе CERT\_FIND\_SUBJECT\_STR, которая определяет, что сертификат ищется по имени (атрибуту CN отличительного имени субъекта). В этом случае параметр pvFindPara является указателем на строку с именем искомого сертификата. Параметр pPrevCertContext – это указатель на структуру типа CERT\_CONTEXT, которая содержит контекст предыдущего сертификата, найденного с этим критерием поиска. Если, как в нашем случае, сертификат с заданным именем будет существовать в единственном экземпляре, этот параметр можно задавать равным NULL.

Функция возвращает указатель на структуру типа CERT\_CONTEXT, которая содержит контекст найденного сертификата или NULL. Пример вызова:

#define MY\_ENCODING\_TYPE (PKCS\_7\_ASN\_ENCODING |\

X509\_ASN\_ENCODING)

TCHAR szCertNameA[] = TEXT(“User\_A”); PCCERT\_CONTEXT pCertA = NULL;

if(!(pCertA = CertFindCertificateInStore(

hStoreMy, MY\_ENCODING\_TYPE, 0, CERT\_FIND\_SUBJECT\_STR,

szCertNameA, NULL)))

{

//Вывод сообщения об ошибке

}

CertGetIssuerCertificateFromStore

Функция позволяет извлечь из заданного хранилища контекст сертификата ЦС, указанного в пользовательском сертификате в качестве издателя. Также функция позволяет произвести простейшую проверку целостности пользовательского сертификата, используя информацию сертификата ЦС. Имеет следующий прототип:

PCCERT\_CONTEXT WINAPI CertGetIssuerCertificateFromStore(

HCERTSTORE hCertStore, PCCERT\_CONTEXT pSubjectContext, PCCERT\_CONTEXT pPrevIssuerContext, DWORD \*pdwFlags );

Параметр hCertStore является дескриптором открытого хранилища, в котором ищется сертификат ЦС. Обычно это «Доверенные корневые центры сертификации» (ROOT). Параметр pSubjectContext – это указатель на структуру типа CERT\_CONTEXT, которая содержит контекст пользовательского сертификата, для которого отыскивается сертификат издателя. Параметр pPrevIssuerContext указывает на контекст предыдущего найденного сертификата этого же ЦС (если их несколько). Если ищется первый (или единственный) сертификат, то этот параметр задается как NULL. Параметр pdwFlags позволяет задать характер осуществляемой проверки пользовательского сертификата. В качестве простейшего варианта можно задать верификацию ЭП в пользовательском сертификате открытым ключом центра сертификации, а также проверку соответствия текущего времени периоду действия сертификата. Для этого данный параметр можно представить комбинацией флагов CERT\_STORE\_SIGNATURE\_FLAG и CERT\_STORE\_TIME\_VALIDITY\_FLAG, объединенных с помощью побитового ИЛИ.

Функция возвращает контекст найденного сертификата или NULL, в случае его отсутствия в указанном хранилище. Если сертификат ЦС найден, то нулевое значение параметра pdwFlags после вызова функции свидетельствует об успешности проверки пользовательского сертификата. Подробную информацию о возвращаемых ненулевых значениях параметра можно получить в соответствующем разделе MSDN.

CertFreeCertificateContext

Функция освобождает контекст ранее найденного в хранилище сертификата. Имеет следующий прототип:

BOOL WINAPI CertFreeCertificateContext(

PCCERT\_CONTEXT pCertContext );

Единственный параметр представляет собой указатель на структуру типа CERT\_CONTEXT, которая содержит контекст сертификата. Функция всегда возвращает ненулевое значение.

CryptSignAndEncryptMessage

Одна из упрощенных функций для создания криптографического сообщения в соответствии со стандартом CMS. Результатом работы функции будет блоб, в котором будет находиться созданное сообщение. Для его создания требуются контексты сертификата отправителя с закрытым ключом и сертификата получателя с открытым ключом (или нескольких сертификатов). В процессе работы функции исходное сообщение хэшируется, подписывается закрытым ключом отправителя, зашифровывается сгенерированным сеансовым ключом. Зашифрованное сообщение также хэшируется, подписывается, после чего ко всей этой информации добавляется зашифрованный открытым ключом получателя сеансовый ключ и создается общий упакованный набор данных. Затем эти упакованные данные вновь хэшируются и подписываются и вместе с электронной подписью составляют результат работы функции – выходной блоб. Функция имеет прототип:

BOOL WINAPI CryptSignAndEncryptMessage(

PCRYPT\_SIGN\_MESSAGE\_PARA pSignPara,

PCRYPT\_ENCRYPT\_MESSAGE\_PARA pEncryptPara, DWORD cRecipientCert,

PCCERT\_CONTEXT rgpRecipientCert[], const BYTE \*pbToBeSignedAndEncrypted, DWORD cbToBeSignedAndEncrypted,

BYTE \*pbSignedAndEncryptedBlob, DWORD \*pcbSignedAndEncryptedBlob );

Параметр pSignPara является указателем на структуру CRYPT\_SIGN\_MESSAGE\_PARA, которая содержит параметры электронной подписи. В MSDN имеется подробное описание данной структуры, мы же приведем пример того, как можно инициализировать данную структуру перед вызовом функции:

CRYPT\_SIGN\_MESSAGE\_PARA SignPara = {

sizeof(CRYPT\_SIGN\_MESSAGE\_PARA) };

SignPara.dwMsgEncodingType = MY\_ENCODING\_TYPE; SignPara.pSigningCert = pCertA; SignPara.HashAlgorithm.pszObjId = szOID\_RSA\_SHA256RSA; SignPara.cMsgCert = 1;

SignPara.rgpMsgCert = &pCertA;

Поле cbSize, определяющее размер структуры в байтах, в данном примере задается при начальной инициализации переменной. Поле dwMsgEncodingType как обычно является результатом побитового ИЛИ между константами X509\_ASN\_ENCODING и PKCS\_7\_ASN\_ENCODING. Поле pSigningCert содержит указатель на контекст сертификата отправителя сообщения. Поле HashAlgorithm является в свою очередь структурой типа CRYPT\_ALGORITHM\_IDENTIFIER, поле которой pszObjId позволяет задать строку с OID комбинации алгоритма подписи, хэширования и шифрования (или предопределенную константу, например szOID\_RSA\_SHA256RSA).

Если сообщение необходимо подписать нескольким пользователям, то массив указателей на контексты их сертификатов передается через поле rgpMsgCert. Поле cMsgCert задает размерность этого массива. В нашем случае эти поля заданы таким образом потому, что сертификат отправителя единственный. Остальные поля структуры остаются нулевыми.

Следующий параметр функции pEncryptPara является указателем на структуру типа CRYPT\_ENCRYPT\_MESSAGE\_PARA, которая содержит параметры шифрования сообщения. Ниже показан пример инициализации данной структуры перед вызовом функции:

CRYPT\_ENCRYPT\_MESSAGE\_PARA EncryptPara = {

sizeof(CRYPT\_ENCRYPT\_MESSAGE\_PARA)

};

EncryptPara.dwMsgEncodingType = MY\_ENCODING\_TYPE; EncryptPara.ContentEncryptionAlgorithm.pszObjId =

szOID\_NIST\_AES128\_CBC;

Назначение поля dwMsgEncodingType аналогично полю в предыдущей структуре. Поле pszObjId вложенной структуры ContentEncryptionAlgorithm позволяет задать OID алгоритма шифрования данных (в данном примере AES-128 в режиме CBC).

Следующая пара параметров функции: cRecipientCert и

rgpRecipientCert по назначению аналогичны полям cMsgCert и

rgpMsgCert структуры CRYPT\_SIGN\_MESSAGE\_PARA, только задают массив указателей на контексты сертификатов получателей, чьими открытыми ключами должны шифроваться сеансовые ключи. В нашем случае сертификат получателя будет единственным, также как и отправителя.

Параметр pbToBeSignedAndEncrypted содержит адрес буфера с исходным сообщением, а параметр cbToBeSignedAndEncrypted задает его размер. Результирующий блоб записывается в буфер, адрес которого задается через параметр pbSignedAndEncryptedBlob, а размер блоба возвращается через параметр pcbSignedAndEncryptedBlob. Если при вызове функции в качестве параметра pbSignedAndEncryptedBlob задать NULL, то через параметр pcbSignedAndEncryptedBlob вернется требуемый размер буфера. Далее буфер такого размера можно создать в динамической памяти и его адрес передать в качестве параметра pbSignedAndEncryptedBlob при повторном (уже результативном) вызове функции.

Функция возвращает значение TRUE при успешном завершении и FALSE при неудаче. В последнем случае код ошибки можно определить с помощью вызова функции GetLastError.

CryptDecryptAndVerifyMessageSignature

Функция расшифровывает криптографическое сообщение, созданное предыдущей функцией и верифицирует входящую в его состав электронную подпись. Необходимые для этого сертификаты функция самостоятельно извлекает из указанных хранилищ. Имеет прототип:

BOOL WINAPI CryptDecryptAndVerifyMessageSignature( PCRYPT\_DECRYPT\_MESSAGE\_PARA pDecryptPara, PCRYPT\_VERIFY\_MESSAGE\_PARA pVerifyPara, DWORD dwSignerIndex,

const BYTE \*pbEncryptedBlob, DWORD cbEncryptedBlob,

BYTE \*pbDecrypted, DWORD \*pcbDecrypted,

PCCERT\_CONTEXT \*ppXchgCert, PCCERT\_CONTEXT \*ppSignerCert );

Параметр pDecryptPara указывает на структуру типа CRYPT\_DECRYPT\_MESSAGE\_PARA, которая несет информацию, необходимую для расшифрования сообщения. Приведем пример инициализации такой структуры перед вызовом функции:

CRYPT\_DECRYPT\_MESSAGE\_PARA DecryptPara = {

sizeof(CRYPT\_DECRYPT\_MESSAGE\_PARA)

};

DecryptPara.dwMsgAndCertEncodingType = MY\_ENCODING\_TYPE; DecryptPara.cCertStore = 1;

DecryptPara.rghCertStore = &hStoreMy;

Поле dwMsgAndCertEncodingType инициализируется той же константой, что и аналогичные поля в рассмотренных ранее структурах. Поле rghCertStore задает массив дескрипторов хранилищ, открытых ранее функцией CertOpenSystemStore. В нашем случае используется одно хранилище (MY), поэтому поле инициализируется адресом его дескриптора, а поле cCertStore получает значение 1.

Параметр pVerifyPara является указателем, содержащим адрес структуры типа CRYPT\_VERIFY\_MESSAGE\_PARA с информацией, необходимой для верификации ЭП в криптографическом сообщении. Пример инициализации такой структуры:

CRYPT\_VERIFY\_MESSAGE\_PARA VerifyPara = {

sizeof(CRYPT\_VERIFY\_MESSAGE\_PARA) };

VerifyPara.dwMsgAndCertEncodingType = MY\_ENCODING\_TYPE;

В данном случае достаточно инициализировать только лишь поля cbSize и dwMsgAndCertEncodingType, назначение которых аналогично одноименным полям в предыдущей структуре.

Параметр dwSignerIndex используется в случае, если сообщение подписывало несколько пользователей и в нашем случае должен быть равен нулю. Параметр pbEncryptedBlob содержит адрес буфера с загруженным блобом криптографического сообщения, а параметр cbEncryptedBlob задает его размер. Параметр pbDecrypted задает адрес буфера, в который функция запишет расшифрованное сообщение, а параметр pcbDecrypted определяет его размер. Если при вызове функции параметр pbDecrypted задать равным NULL, то через параметр pcbDecrypted вернется требуемый размер буфера.

Параметры ppXchgCert и ppSignerCert после вызова функции должны указывать на контексты сертификатов, содержащих соответственно, закрытый ключ получателя сообщения и открытый ключ отправителя (в нашем случае для простоты это будет тот же сертификат с закрытым ключом, который использовался для создания сообщения). В дальнейшем эти контексты нужно будет освободить с помощью функции CertFreeCertificateContext. Если же вызов функции закончится неудачей вследствие невозможности расшифровать сообщение, то эти параметры будут установлены в значение NULL.

Функция возвращает значение TRUE при успешном завершении и FALSE при неудаче. В последнем случае код ошибки можно определить с помощью вызова функции GetLastError.

СОДЕРЖАНИЕ РАБОТЫ

1. С помощью криптографического пакета OpenSSL создать:

 ключевую пару алгоритма RSA с длиной ключа 2048 бит и соответствующий ей самоподписанный сертификат центра сертификации;

 две ключевые пары алгоритма RSA с длиной ключа 2048 бит и соответствующие им сертификаты в формате PKCS#12 для двух пользователей – участников процесса обмена криптографическими сообщениями. Сертификаты должны быть подписаны закрытым ключом центра сертификации.

2. Установить в системе созданные сертификаты. В отчет внести последовательность команд OpenSSL, использованных для создания сертификатов центра сертификации и пользователей.

3. Разработать на языке программирования C/C++ с использованием средств криптографического интерфейса Microsoft CryptoAPI консольное или оконное приложение, выполняющее создание криптографического сообщения по стандарту CMS из указанного пользователем файла и дальнейшего его расшифрования. Криптографическое сообщение должно содержать данные, зашифрованные алгоритмом AES-128 в режиме CBC и электронную подпись, созданную с помощью алгоритма RSA. Приложение должно предлагать пользователю перечень имен субъектов сертификатов, установленных в хранилище «Личное», и принимать его выбор имен отправителя и получателя криптографического сообщения. Перед созданием сообщения необходимо верифицировать ЭП в составе выбранных сертификатов и проверить соответствие текущей даты периоду, заданному в их составе. Созданное криптографическое сообщение необходимо выгружать в указанный пользователем файл и загружать из него в память для расшифрования. Расшифрованное сообщение также необходимо выгружать в файл, указанный пользователем.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего используются сертификаты открытых ключей X.509?

2. Что такое инфраструктура открытых ключей (PKI)? Какие варианты архитектуры PKI вы знаете?

3. Какова структура сертификата X.509?

4. Как сертификаты X.509 хранятся в запоминающих устройствах? Какие форматы сертификатов вы знаете?

5. Что такое поля расширений в составе сертификата X.509?

6. Как OpenSSL настраивается для работы тестового центра сертификации?

7. Какие команды OpenSSL используются для создания сертификатов?

8. Как установить созданный сертификат в системе?

9. Какие в ОС Windows имеются средства для управления установленными сертификатами?

10. Что определяют спецификация PKCS#7 и стандарт CMS?

11. Какие функции Microsoft CryptoAPI для управления хранилищами сертификатов вы знаете?

12. Какие функции Microsoft CryptoAPI для работы с сертификатами вы знаете?

13. Как определить имена всех сертификатов в хранилище?

14. Как верифицировать сертификат?

15. Какие функции Microsoft CryptoAPI поддержки криптографических сообщений вы знаете?

16. Какие структуры данных подготавливаются перед вызовом функции, создающей криптографическое сообщение?

17. Какие структуры данных подготавливаются перед вызовом функции, расшифровывающей криптографическое сообщение?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 12 РЕАЛИЗАЦИЯ ЗАЩИЩЕННОЙ ПЕРЕДАЧИ ДАННЫХ ПО ПРОТОКОЛУ TLS СРЕДСТВАМИ КРИПТОГРАФИЧЕСКОГО ПАКЕТА OPENSSL**

Цель работы: ознакомиться со способами организации защищенных TLS-соединений средствами криптографического пакета OpenSSL, а также получить навыки в создании простейших клиентских и серверных приложений в среде Visual Studio.

ОСНОВНЫЕ ПОНЯТИЯ

Создание сертификатов открытых ключей алгоритма электронной подписи ГОСТ Р 34.10-2001

В прошлой лабораторной работе был рассмотрен процесс создания с помощью OpenSSL сертификатов X.509, содержащих ключи асимметричного алгоритма RSA, которые использовались как для обмена сеансовыми ключами, так и для создания и верификации электронной подписи (ЭП). То, что для этих целей не использовались отечественные криптоалгоритмы, было обусловлено тем, что в составе Windows по умолчанию отсутствуют криптопровайдеры с их поддержкой. В данной же работе мы будем применять только средства пакета OpenSSL, поэтому для создания защищенных соединений будем использовать сертификаты ключей отечественных криптоалгоритмов, и прежде всего алгоритма создания и проверки электронной подписи ГОСТ Р 34.10-2001.

Протокол TLS (Transport Layer Security), как и его предшественник SSL (Secure Sockets Layer) предполагают установление защищенного соединения с обязательной аутентификацией сервера и опциональной аутентификацией клиента. В данной работе будет рассмотрен простейший (и, в общем-то, самый распространенный) вариант с односторонней аутентификацией сервера. Причем под сервером в данном случае мы понимаем компьютер (узел в сети), к которому клиент хочет подключиться и передать какую-либо конфиденциальную информацию. Для аутентификации и последующего согласования сеансового ключа серверу необходимы собственные сертификат и закрытый ключ, а клиенту – сертификат (или доверительная цепочка сертификатов) центра сертификации,

который сертифицировал сервер. Если клиент доверяет сертификату центра сертификации, то он может проверить валидность сертификата сервера. После этой проверки клиент может быть уверен, что обменивается данными именно с тем узлом, за который себя выдает сервер (в реальности если сервер является веб-узлом, то атрибут CN его сертификата задается равным доменному имени).

Для реализации передачи данных по протоколу TLS сгенерируем самоподписанный сертификат центра сертификации и с его помощью создадим сертификат сервера. Будем предполагать, что пакет OpenSSL сконфигурирован так, как описывалось в предыдущих лабораторных работах. Запустим файл openssl.exe и для создания самоподписанного сертификата выполним команду:

req -x509 -newkey gost2001 -pkeyopt paramset:A -keyout ca\_gost\_key.pem -out ca\_gost\_cert.pem -days 730

Данная команда создаст новую ключевую пару алгоритма ГОСТ Р 34.10-2001 с набором параметров "A" (с сохранением в файле ca\_gost\_key.pem) и сертификат со сроком действия в 2 года. После запуска команды необходимо ввести пароль для шифрования закрытого ключа и данные для сертификата. Большинство параметров отличительного имени принимаем заданными по умолчанию, а атрибут CN определяем как CA\_GOST. Теперь создадим запрос на получение сертификата сервера также с генерацией ключевой пары:

req -newkey gost2001 -pkeyopt paramset:A -keyout serv\_key.pem -out serv\_req.pem

При создании запроса также надо будет ввести пароль и данные субъекта. В качестве атрибута CN отличительного имени субъекта укажем строку Server\_TLS. Теперь на основании созданного запроса нужно выпустить сертификат:

ca -in serv\_req.pem -out serv\_cert.pem -keyfile ca\_gost\_key.pem -cert ca\_gost\_cert.pem

Созданный сертификат и его закрытый ключ далее будут использованы для организации TLS-соединения между сервером и клиентом.

Настройка проекта Visual Studio и инициализация библиотеки OpenSSL для создания TLS-соединений

Для начала необходимо определиться с тем, какие статические и динамические библиотеки нам понадобятся для создания клиентского

и серверного приложений в среде Visual Studio. Из стандартных библиотек Windows нам дополнительно понадобится файл ws2\_32.dll, который содержит реализацию WinSock API – интерфейса для созданий сетевых приложений с обменом данными через сокеты. Соответственно, будем подключать статическую библиотеку импорта ws2\_32.lib, указывая ее имя в свойствах проекта Visual C++ или в директиве:

#pragma comment(lib, " ws2\_32.lib")

Аналогично подключим статические библиотеки OpenSSL libeay32.lib и ssleay32.lib. Соответствующие им динамические библиотеки при запуске проекта на выполнение должны находиться в каталоге решения или системном каталоге, содержащем dll-файлы установленных приложений. Поскольку наши приложения (и клиентское и серверное) в процессе создания TLS-соединения будут использовать отечественные криптоалгоритмы, то нам также понадобится файл gost.dll, который должен находиться в каталоге, определенном в секции [gost\_section] конфигурационного файла openssl.cfg (см. лабораторную работу №4).

Для получения прототипов нужных нам функций достаточно включить в текст программы заголовочные файлы: WinSock2.h, openssl/err.h, openssl/ssl.h.

Также нужно определиться с тем, как в дальнейшем для проверки работоспособности запустить скомпилированное клиентское или серверное приложения на компьютере, не содержащем среды разработки и установленного пакета OpenSSL. Для этого необходимо, чтобы в одной папке с файлом приложения (или в системном каталоге) находились все используемые динамические библиотеки. Также необходимо скопировать (можно в эту папку или в другую, по желанию) конфигурационный файл openssl.cfg. В самом файле в секции [gost\_section] нужно прописать реальный путь (!) к gost.dll. Далее необходимо создать переменную среды OPENSSL\_CONF, содержащую путь к этому конфигурационному файлу. После выполнения этих действий, приложение, выполняя при запуске начальную инициализацию библиотеки OpenSSL, сможет загрузить модуль gost. Также нужно отметить, что при запуске клиентского или серверного приложения, возможно, придется временно отключать брандмауэр Windows или антивирусной программы, так как он может блокировать передачу данных.

Для инициализации библиотеки традиционно будем осуществлять вызов функции OPENSSL\_add\_all\_algorithms\_conf (см. лабораторную

работу №4). Причем вызвать эту функцию нужно до вызова основной инициализирующей функции (см. ниже), чтобы шифрсьюты с поддержкой алгоритмов шифрования и хэширования ГОСТ попали в число доступных при установлении соединения.

Следующим шагом при инициализации библиотеки является вызов функции SSL\_library\_init, имеющей следующий прототип (находится в файле openssl/ssl.h):

int SSL\_library\_init(void);

Функция регистрирует все доступные шифрсьюты и возвращает 1 в случае успешного завершения.

Инициализация таблицы текстовых сообщений об ошибках осуществляется вызовом функции SSL\_load\_error\_strings, имеющей следующий прототип:

void SSL\_load\_error\_strings(void);

Пример получения сообщения о произошедшей ошибке будет приведен ниже.

Также в начале программы можно инициализировать интерфейс WinSock API с помощью функции WSAStartup, так как TLS- соединение будет создаваться на базе уже существующего сетевого соединения (сокета), поддерживающего передачу данных в соответствие с протоколом TCP.

Примечание. В данной работе не будут рассматриваться сигнатура и принцип действия используемых функций интерфейса WinSock API. Получить о них подробную информацию можно, например, в соответствующем разделе MSDN (https://msdn.microsoft.com/en- us/library/windows/desktop/ms741394%28v=vs.85%29.aspx).

Структура простейшего клиентского приложения, передающего и принимающего данные по протоколу TLS

После первоначальной инициализации клиентское приложение должно создать защищенное соединение с сервером и организовать передачу и прием данных. Для этого необходимо выполнить следующие действия:

1. Создать контекст TLS – переменную типа SSL\_CTX (структура, определенная в файле openssl/ssl.h). Установить нужные параметры контекста.

2. Для созданного контекста задать параметры хранилища доверенных сертификатов центров сертификации, используемых при верификации сертификата сервера.

3. Создать объект TLS-соединения – переменную типа SSL (тип также определен в файле openssl/ssl.h).

4. С помощью функций интерфейса WinSock API (socket, bind, connect) открыть обычный сокет и установить логическое соединение по протоколу TCP с заданным сервером (к этому моменту сервер уже должен «слушать» указанный порт на предмет входящих клиентских соединений).

5. Установить полученный на предыдущем шаге дескриптор сокета в объект TLS-соединения, созданный в п. 3.

6. Инициировать установление TLS-соединения (запуск TLS Handshake Protocol – протокола «рукопожатия» или

«хендшейка», целью которого является аутентификация сервера, обмен информацией о применяемых сторонами алгоритмах шифрования и хэширования, а также согласование ключевого материала).

7. В случае успешного завершения хендшейка, организовать передачу и прием необходимых данных. По окончании передачи разорвать TLS-соединение и закрыть сокет.

8. Освободить объект TLS-соединения и контекст TLS.

Далее будут рассмотрены функции библиотеки OpenSSL, необходимые для выполнения этих операций.

Функция SSL\_CTX\_new создает новый контекст TLS. Имеет следующий прототип:

SSL\_CTX \*SSL\_CTX\_new(const SSL\_METHOD \*meth);

Параметр meth указывает, какие версии протокола TLS (или SSL) будут поддерживаться. Существует ряд функций, которые возвращают константный указатель на структуру SSL\_METHOD. В данной работе и в клиентском и в серверном приложении будем использовать функцию:

const SSL\_METHOD \*TLSv1\_method(void);

Для созданного контекста можно устанавливать ряд параметров с помощью соответствующих функций и их опций. В нашем случае можно установить один параметр с помощью функции:

long SSL\_CTX\_set\_mode(SSL\_CTX \*ctx, long mode);

При вызове функции зададим параметр mode равным константе

SSL\_MODE\_AUTO\_RETRY. В нашем случае клиентское приложение

для простоты будет устанавливать блокирующее соединение с сервером. Если этот параметр установлен, то при блокировании транспортного сокета, на базе которого устанавливается TLS- соединение, функции чтения и записи не будут возвращать ошибку и, прозрачно для приложения, будет проводиться повторная процедура хендшейка.

Ниже показан пример создания контекста TLS и установления данного параметра:

SSL\_CTX \*ssl\_ctx = SSL\_CTX\_new(TLSv1\_method()); SSL\_CTX\_set\_mode(ssl\_ctx, SSL\_MODE\_AUTO\_RETRY);

Для проверки сертификата, переданного клиенту сервером, необходимо установить параметры хранилища доверенных сертификатов, которое может представлять собой папку или файл на диске. Путь к ним можно задать с помощью функции:

int SSL\_CTX\_load\_verify\_locations(SSL\_CTX \*ctx,

const char \*CAfile, const char \*CApath);

Параметр ctx является указателем на созданный ранее контекст TLS. Параметр CAfile содержит имя файла, в котором находится сертификат корневого центра сертификации или последовательно перечисленный набор сертификатов в PEM-формате, входящих в доверительную цепочку (сюда же могут быть добавлены списки отозванных сертификатов CRL). В нашем случае для аутентификации сервера (верификации переданного им сертификата) будет использоваться единственный сертификат корневого центра сертификации (ранее сгенерированный и сохраненный в файле ca\_gost\_cert.pem). Строку с его именем необходимо передать в качестве параметра CAfile. В этом случае в качестве параметра CApath передается значение NULL. Функция возвращает 1 в случае успешного завершения.

Создание объекта TLS-соединения можно выполнить с помощью функции:

SSL \*SSL\_new(SSL\_CTX \*ctx);

Для заданного контекста TLS создается переменная типа SSL, указатель на которую возвращается функцией. В случае ошибки возвращается значение NULL.

После того, как с помощью функций интерфейса WinSock API будет открыт обычный сокет и установлено логическое соединение по протоколу TCP с сервером по заданным IP-адресу и номеру порта,

необходимо установить дескриптор сокета в объект TLS-соединения вызовом функции:

int SSL\_set\_fd(SSL \*s, int fd);

Параметр s указывает на ранее созданный объект TLS-соединения, а fd содержит дескриптор открытого сокета, связанного с TCP- соединением. Функция возвращает 1 при успешном завершении.

После установки дескриптора сокета в объект TLS-соединения можно запускать процедуру хендшейка с помощью функции:

int SSL\_connect(SSL \*ssl);

Во время работы данной функции происходит, в том числе, верификация сертификата, переданного сервером, с помощью содержимого хранилища доверенных сертификатов, чьи параметры были установлены для контекста TLS ранее с помощью вызова функции SSL\_CTX\_load\_verify\_locations. В случае удачного завершения хендшейка функция возвращает 1. Нулевое или отрицательное значение свидетельствует об ошибке. Ниже показан пример получения текстового описания ошибки в случае неудачи при установлении соединения с сервером. Код ошибки можно получить с помощью вызова функции SSL\_get\_error.

int nRet;

if ((nRet=SSL\_connect(ssl))<=0)

{

char buffer[500]; ERR\_error\_string(SSL\_get\_error(ssl, nRet), buffer); printf("Ошибка: %s\n", buffer);

}

После того, как соединение установлено, клиент может передавать данные серверу и получать их от него. И клиент и сервер используют для этого одни и те же функции. Для чтения данных из TLS- соединения используется функция:

int SSL\_read(SSL \*ssl, void \*buf, int num);

Из указанного соединения (параметр ssl) считываются данные объемом не более num байт и помещаются в буфер buf. Функция возвращает количество реально прочитанных из соединения байт. Нулевое или отрицательное возвращаемое значение свидетельствуют об ошибке, код которой можно получить с помощью функции SSL\_get\_error.

Для записи данных используется функция:

int SSL\_write(SSL \*ssl, const void \*buf, int num);

Данные объемом num байт считываются из буфера buf и записываются в соединение ssl. Функция возвращает количество реально записанных байт. Диагностирование ошибки осуществляется также как и у функции SSL\_read.

Для того чтобы разорвать TLS-соединение используется функция:

int SSL\_shutdown(SSL \*s);

Функция отправляет соответствующее сообщение другой стороне о закрытии соединения. В случае успешной отправки сообщения и получения ответа от другой стороны функция вернет 1. Это будет свидетельствовать, что обе стороны выполнили данную функцию. Если ответ не получен, то вернется значение нуля. В этом случае можно повторно выполнять функцию до получения 1, что будет означать закрытие соединения другой стороной. Функция вернет значение -1 в случае ошибки, код которой можно будет извлечь с помощью вызова SSL\_get\_error.

После разрыва TLS-соединения приложение может закрывать TCP- соединение и сокет с помощью функций интерфейса WinSock API shutdown и closesocket соответственно. Далее освобождается объект TLS-соединения с помощью вызова функции:

void SSL\_free(SSL \*ssl);

Данная функция уменьшает на единицу количество ссылок на указанный объект и уничтожает его после обнуления этого количества. Освободить контекст TLS можно вызовом функции:

void SSL\_CTX\_free(SSL\_CTX \*);

Принцип работы данной функции аналогичен функции SSL\_free.

Структура простейшего серверного приложения, принимающего и передающего данные по протоколу TLS

Для работы простейшего серверного приложения необходимо реализовать следующий алгоритм:

1. Создать контекст TLS и установить его параметры аналогично клиентскому приложению.

2. Установить в контекст сертификат сервера и его закрытый ключ.

3. С помощью функций интерфейса WinSock API (socket, bind) открыть обычный сокет для установления соединения по

протоколу TCP. Перевести созданный сокет в режим прослушивания клиентских соединений вызовом функции listen.

4. После успешного завершения функции listen, вызвать функцию accept, которая ожидает появление входящего соединения и возвращает новый подключенный сокет. Создать новый объект TLS-соединения и установить в него (аналогично клиентскому приложению) возвращенный функцией accept дескриптор сокета.

5. Перевести TLS-соединение в режим ожидания инициирования хендшейка клиентом. После его успешного завершения организовать прием и передачу необходимых данных. По окончании разорвать TLS-соединение и закрыть сокет.

6. Освободить объект TLS-соединения и контекст TLS.

Для того чтобы установить сертификат сервера в контекст TLS можно использовать функцию:

int SSL\_CTX\_use\_certificate\_file(SSL\_CTX \*ctx,

const char \*file, int type);

В качестве параметра file нужно передать строку с именем файла, содержащего сертификат сервера (в нашем случае serv\_cert.pem). Для того чтобы показать, что файл содержит сертификат в формате PEM, параметр type нужно задать равным константе SSL\_FILETYPE\_PEM. Функция возвращает значение 1 в случае удачного завершения и ноль в случае ошибки.

Установить в контекст сертификата закрытый ключ можно с помощью функции:

int SSL\_CTX\_use\_PrivateKey\_file(SSL\_CTX \*ctx,

const char \*file, int type);

Назначение параметров данной функции аналогично таковым у рассмотренной выше функции SSL\_CTX\_use\_certificate\_file. При загрузке закрытого ключа (в нашем случае файл serv\_key.pem) функция потребует ввести в консоли пароль. Если же пароль необходимо задавать в тексте программы, то перед вызовом данной функции можно выполнить функцию:

void SSL\_CTX\_set\_default\_passwd\_cb\_userdata(SSL\_CTX \*ctx,

void \*u);

Эта функция задает пароль, который будет использоваться при доступе к закрытому ключу функцией SSL\_CTX\_use\_PrivateKey\_file. В качестве параметра u передается строка с паролем максимальной длины 255 символов.

После установления с клиентом TCP-соединения и создания объекта TLS-соединения, его необходимо перевести в режим ожидания запроса клиента на проведение хендшейка. Это делается с помощью функции:

int SSL\_accept(SSL \*ssl);

Функция вернет значение 1 при успешном завершении и нулевое или отрицательное значение в случае ошибки (пример того, как получить информацию об ошибке был рассмотрен при описании функции SSL\_connect).

После успешного установления соединения сервер может принимать и передавать информацию с помощью ранее описанных функций SSL\_read и SSL\_write. По окончании обмена данными производятся те же действия по разрыву TLS и TCP соединений, закрытию сокета, освобождению объекта TLS-соединения и контекста TLS, что и в клиентском приложении.

СОДЕРЖАНИЕ РАБОТЫ

1. С помощью криптографического пакета OpenSSL создать:

 ключевую пару алгоритма ГОСТ Р 34.10-2001 с набором параметров "A" (RFC 4357) и соответствующий ей самоподписанный сертификат центра сертификации;

 ключевую пару алгоритма ГОСТ Р 34.10-2001 с набором параметров "A" и соответствующий ей сертификат в PEM- формате для сервера. Сертификат должен быть подписан закрытым ключом центра сертификации.

2. Разработать на языке программирования C/C++ с использованием средств криптографического пакета OpenSSL консольные клиентское и серверное приложения (или одно приложение, имеющее два режима), предназначенные для создания блокирующего соединения по протоколу TLS и выполняющие обмен сообщениями. Клиентское приложение должно запрашивать у пользователя IP-адрес в десятично- точечной форме. Номер порта (произвольное значение больше 1023) определяется в приложениях заранее. В качестве очередного сообщения должна выступать символьная строка заранее оговоренной максимальной длины, которую

пользователь клиентского приложения вводит в консоли, после чего она передается серверу. Сервер, получив строку, выводит ее в консоль и возвращает ее клиенту. Клиент также для подтверждения приема выводит строку в консоль. Строка должна передаваться вместе с ограничивающим ее длину нуль- символом (0). Передача клиентом строки нулевой длины (то есть состоящей только из одного нуль-символа) будет означать окончание сеанса связи.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как в протоколе TLS осуществляется аутентификация сервера и клиента?

2. Как с помощью криптографического пакета OpenSSL осуществить генерацию ключевой пары алгоритма ГОСТ 34.10-2001 и создание самоподписанного сертификата?

3. Что собой представляет обобщенный алгоритм работы клиентского приложения, передающего и принимающего данные по протоколу TLS?

4. Какие функции WinSock API используются для открытия и закрытия сокетов, создания и разрыва TCP-соединений?

5. Как для клиентского приложения установить параметры хранилища доверенных сертификатов?

6. Что такое TLS Handshake Protocol?

7. Какие функции OpenSSL используются для создания и установки параметров контекста TLS?

8. Как создать объект TLS-соединения и связать его с сокетом, поддерживающим TCP-соединение?

9. Как разорвать TLS-соединение и освободить его объект и контекст TLS?

10. Как клиентское приложение может инициировать процедуру хендшейка?

11. Какие функции OpenSSL используются для передачи и приема данных по протоколу TLS?

12. Что собой представляет обобщенный алгоритм работы серверного приложения, передающего и принимающего данные по протоколу TLS?

13. Как установить в контекст TLS серверного приложения сертификат сервера и его закрытый ключ?

14. Как перевести серверное приложение в режим ожидания запроса клиента на проведение хендшейка?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 13. ШИФРОВАНИЕ СООБЩЕНИЙ С ПОМОЩЬЮ ШИФРОВ ЗАМЕНЫ КРИПТОГРАФИЧЕСКИМИ ФУНКЦИЯМИ EXCEL**

Цель: Освоение методов простейшего шифрования. Задачи,

1. Ознакомиться с теорией.

2. Разработать, аналогично примеру, программу реализации простейшего шифра замены (на примере шифра Цезаря).

3. Зашифровать текст.

4. Написать программу расшифровки зашифрованного сообщения.

5. Выполнить расшифровку.

6. Закрепить навыки на примере шифра Вижинера.

7. Оформить отчет.

Средство реализации: среда программирования Delphi, MS Excel

Литература 2, 3, 6, 9, 11, 13, 14,15, 16, 17

Теоретические основы

Цезарь (I в. до н.э.) заменял буквы открытого текста буквами шифралфавита, который получался циклическим сдвигом букв алфавита открытого текста на три позиции вправо. В общем случае величина сдвига L, на который сдвигаются буквы шифралфавита, может быть любой в интервале L = 1..25.

Если пронумеровать буквы алфавита в порядке 0, 1, 2, …, |A|-1, ввести имена M - номер буквы алфавита, C – номер соответствующей буквы шифроалфавита, L - величина сдвига, |A| – размер алфавита (для английского алфавита |A| = 26), то выполняется соотношение:

C = (M + L) mod |A|,

поэтому вместо таблицы подстановки в программной реализации шифра Цезаря можно воспользоваться этой простой математической формулой.

Дешифрование криптограммы выполняется с помощью аналогичной формулы, где вместо операции сложения берется операция вычитания:

M = (C - L) mod |A|

Шифр Цезаря легко взламывается перебором значений L = 1..25, пока не будет получен осмысленный текст, либо методом частотного анализа, как частный случай одноалфавитных шифров.

В прилагаемом к лабораторной работе файле Excel используются две пользовательские функции поддержки шифра Цезаря: PlainEngTextA(m) и EncryptEngTextA(m,L).

Функция PlainEngTextA(m) преобразует строчные буквы исходного английского текста в заглавные, а также ищет в тексте все неалфавитные символы и заменяет их одиночными пробелами.

Входной параметр m функции - ячейка рабочего листа Excel, содержащая исходный текст, предназначенный для шифрования.

Программный код функции PlainEngTextA(m) имеет вид: Function PlainEngTextA(m As Range) As String

Const firstAsc = 65 ' - код первой буквы английского алфавита

Const lastAsc = firstAsc + 26 ' - код последней буквы английского алфавита

Dim msg As String ' - исходный английский текст Dim mLen As Long ' - длина исходного текста Dim s As String ' – открытый английский текст Dim sym As String ' - символ открытого текста

Dim i As Long ' - счетчик циклов

Dim cnt As Long ' - счетчик неалфавитных символов

msg = Ucase(Trim(m)) mLen = Len(msg)

cnt = 0 s = ""

For i = 1 To mLen sym = Mid(msg, i, 1)

If (Asc(sym) >= firstAsc) And (Asc(sym) < lastAsc) Then cnt = 0

s = s & sym Else

cnt = cnt + 1

If cnt = 1 Then s = s & " " End If

Next PlainEngTextA = s End Function

Функция EncryptEngTextA(m,L) шифрует открытый текст шифром Цезаря.

Входной параметр m функции - ячейка рабочего листа Excel, содержащая открытый текст, полученный из исходного текста с помощью функции PlainEngTextA(m,L).

Входной параметр L функции - ячейка рабочего листа Excel, содержащая величину сдвига.

Программный код функции EncryptRusTextA(m,L) имеет вид:

Function EncryptEngTextA(m As Range, L As Long) As String

Const firstAsc = 65 ' - ASCII-код первой буквы английского алфавита

Const aLen = 26 ' - размер английского алфавита Dim msg As String ' - открытый английский текст Dim mLen As Long ' - длина открытого текста

Dim cip As String ' – криптограмма открытого текста

Dim sym As String ' - символ открытого текста

Dim i As Long ' - счетчик циклов

Dim code As Long ' - код символа криптограммы

msg = Trim(m) mLen = Len(msg) cip = ""

For i = 1 To mLen sym = Mid(msg, i, 1)

If sym = " " Then cip = cip & " "

Else

code = Asc(sym) + L - firstAsc

cip = cip & Chr(firstAsc + code - aLen \* Int(code / aLen)) End If

Next

EncryptEngTextA = cip End Function

Порядок шифрования исходного текста и дешифрования полученной криптограммы с помощью этих функций показан на рис.1.

Шифр Виженера

Важное усовершенствование многоалфавитных систем, состоящее в использовании в качестве ключа текста самого сообщения или шифрованного текста, принадлежит Виженеру. В простейшем случае за основу бралась таблица Тритемия с добавленными к ней в качестве первой строки и первого столбца английскими алфавитами. Позже такая таблица стала называться таблицей Виженера. Первая строка таблицы служит алфавитом открытого текста, а первый столбец — алфавитом ключа.

Отметим, что в общем случае таблица Виженера состоит из циклически сдвигаемых произвольных смешанных алфавитов.

Если пронумеровать буквы алфавита в порядке 0, 1, 2, …, |A|-1 ввести имена M - номер буквы алфавита, C – номер соответствующей буквы шифроалфавита, K – номер соответствующей буквы ключа, |A| – длина алфавита (для английского алфавита

|A| = 26), то выполняется соотношение: C = (M + K) mod |A|,

поэтому вместо громоздкой таблицы подстановки в программной реализации шифра Виженера можно воспользоваться этой простой математической формулой.

Дешифрование криптограммы выполняется с помощью аналогичной формулы, где вместо операции сложения берется операция вычитания:

M = (C - K) mod |A|

Шифр Виженера был взломан известным английским математиком Бэббиджем. Для этого Бэббидж начал с отыскивания в криптограмме одинаковых последовательностей из 3- х, 4-х или 5-ти букв, предполагая, что они представляют собой фрагменты одинакового текста, зашифрованные одинаковыми буквами ключа. Затем нашел наименьший общий делитель расстояний между одинаковыми последовательностями букв, который, очевидно, должен равняться длине ключа Lk.

Дальнейший взлом шифра выполнялся применением частотного анализа Lk раз, где для каждого анализа выбираются последовательности букв криптограммы, расстояние между которыми равно Lk.

В прилагаемом к лабораторной работе файле Excel используются две

пользовательские функции поддержки шифра Виженера: PlainRusTextB(m) и

EncryptRusTextB(m,L).

Функция PlainRusTextB(m) преобразует строчные буквы исходного русского текста в заглавные, убирает из текста пробелы между словами, цифры и знаки пунктуации. Чтобы функция работала правильно, в исходном тексте вместо буквы ё нужно использовать букву е.

Входной параметр m функции - ячейка рабочего листа Excel, содержащая исходный текст, предназначенный для шифрования.

Программный код функции PlainRusTextB(m) имеет вид: Function PlainRusTextB(m As Range) As String

Const firstCode = 192 ' - ASCII-код первого символа алфавита

Const lastCode = 223 ' - ASCII-код последнего символа алфавита

Dim msg As String ' - исходный текст

Dim mLen As Long ' - длина исходного текста

Dim s As String ' - открытый текст

Dim sym As String ' - символ открытого текста Dim code As Byte ' - ASCII-код этого символа Dim i As Long ' - счетчик циклов

msg = Ucase(Trim(m)) mLen = Len(msg)

cip = ""

For i = 1 To mLen sym = Mid(msg, i, 1) code = Asc(sym)

If (code >= firstCode) And (code <= lastCode) Then s = s & sym Next

PlainRusTextB = s End Function

Функция EncryptRusTextB(m,k) шифрует открытый текст шифром Виженера.

Входной параметр m функции - ячейка рабочего листа Excel, содержащая открытый текст, полученный из исходного русского текста с помощью функции PlainRusTextB(m,k).

Входной параметр k функции - ячейка рабочего листа Excel, содержащая ключ шифрования, который представляет собой последовательность неповторяющихся русских букв, не содержащую пробелов и знаков пунктуации.

Программный код функции EncryptRusTextB(m,k) имеет вид:

Function EncryptRusTextB(m As Range, k As Range) As String Const firstCode = 192 ' - ASCII-код первого символа алфавита Const lastCode = 223 ' - ASCII-код последнего символа алфавита Dim aLen As Long ' - число символов алфавита

Dim msg As String ' - открытый текст

Dim mLen As Long ' - длина открытого текста

Dim cip As String ' - шифротекст

Dim sym As String ' - символ открытого текста

Dim kym As String ' - символ ключа

Dim key As String ' - ключ

Dim kLen As Long ' - длина ключа

Dim i As Long ' - счетчик циклов

Dim ki As Long ' - счетчик символов ключа

Dim vi As Long ' - номер строки таблицы Виженера

Dim vj As Long ' - номер столбца таблицы Виженера

aLen = lastCode - firstCode + 1 msg = Trim(m)

mLen = Len(msg) key = Trim(k) kLen = Len(key) cip = ""

For i = 1 To mLen sym = Mid(msg, i, 1)

vj = Asc(sym) - firstCode

ki = i - Int((i - 1) / kLen) \* kLen kym = Mid(key, ki, 1)

vi = Asc(kym) - firstCode

cip = cip & Chr(firstCode + vj + vi - Int((vi + vj) / aLen) \* aLen) Next

EncryptRusTextB = cip End Function

Порядок шифрования исходного русского текста и дешифрования полученной криптограммы с помощью этих функций аналогичен порядку, показанному на рис.1.

Задание для выполнения

1. Изучите программный код пользовательских функций PlainEngTextA() и EncryptEngTextA() и научитесь использовать их на рабочем листе Excel для шифрования английских текстов шифром Цезаря.

2. Напишите программный код пользовательской функции DecryptEngTextA(), предназначенной для дешифрования английских криптограмм, полученных с помощью шифра Цезаря. Очевидно, что эта функция является обратной функции EncryptEngTextA().

3. Напишите программный код пользовательских функций PlainRusTextA(), EncryptRusTextA() и DecryptRusTextA() поддержки шифрования и дешифрования русских текстов шифром Цезаря. Зашифруйте и дешифруйте с их помощью какой-нибудь русский текст.

4. Изучите программный код пользовательских функций PlainRusTextB() и EncryptRusTextB() и научитесь использовать их на рабочем листе Excel для шифрования русских текстов шифром Виженера.

5. Напишите программный код пользовательской функции DecryptRusTextB(), предназначенной для дешифрования русских криптограмм, полученных с помощью шифра Виженера. Очевидно, что эта функция является обратной функции EncryptRusTextB().

6. Напишите программный код пользовательских функций PlainEngTextB(), EncryptEngTextB() и DecryptEngTextB() поддержки шифрования и дешифрования английских текстов шифром Виженера. Зашифруйте и дешифруйте с их помощью какой-нибудь английский текст.

Контрольные вопросы

1. Схематично описать алгоритм Цезаря.

2. Схематично описать алгоритм Вижинера.

3. Какой алгоритм, на ваш взгляд, более эффективен, почему?

Лабораторная работа №2. Криптоаналитический частотный анализ одноалфавитных шифров средствами Ecxel

Цель: освоение базовых навыков криптоанализа

Средство реализации: MS Excel

Литература 2, 3, 6, 9, 11, 13, 14,15, 16, 17

Теоретические основы

Рекомендации по частотному анализу шифротекста

1. Начните с подсчета частоты появления каждой из букв шифротекста. Например, примерно пять букв английского текста должны появляться с частотой менее 1%, и они, вероятно, представляют собой буквы k, q, x и z. Одна из букв должна появляться с частотой более 10%, и это, по-видимому, буква е. Если шифротекст не подчиняется этому распределению частот, то, возможно, открытый текст написан не на английском языке.

2. Если установлена взаимосвязь с английским языком, но открытый текст, как это часто происходит, сразу не появляется, то нужно обратить внимание на пары повторяющихся букв. В английском языке чаще всего повторяются буквы ss, ее, tt, ff, ll, mm и oo. Если в шифротексте имеются повторяющиеся символы, то можно считать, что они представляют собой одну из этих пар.

3. Если в шифротексте имеются пробелы между словами, то определите слова, состоящие из одной, двух и трех букв. Единственными словами в английском языке, состоящими из одной буквы, являются a и I. Наиболее частые двухбуквенные слова - of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am, трехбуквенные слова — the и and.

4. Если удастся, подготовьте таблицу частотности букв для текста, который вы пытаетесь дешифровать. Например, в военных донесениях стремятся опускать местоимения и артикли, и это искажает стандартную частотность букв языка. Если вы знаете, что работаете с военным донесением, то следует использовать таблицу частотности букв, созданную на основе других военных донесений.

5. Одно из самых полезных для криптоаналитика умений — способность распознавать отдельные слова и даже фразы. Так, аль-Халил, один из первых арабских криптоаналитиков, продемонстрировал свои способности, когда взломал греческий шифротекст, предположив, что шифротекст начинается с приветствия «Во имя бога». Выяснив, что это приветствие действительно соответствуют определенному фрагменту шифротекста, он смог раскрыть остальной шифротекст.

6. В некоторых случаях наиболее часто встречающейся буквой в шифротексте может быть E, следующей по частоте появления — Т и т.д. Другими словами, частотность букв в шифротексте совпадает с частотностью букв в языке, и все же шифротекст выглядит тарабарщиной. В этом случае вы столкнулись не с шифром замены, а с шифром перестановки, т.е. все буквы в шифротексте остались теми же самыми, но находятся не на своих местах.

Предположим, что нужно дешифровать следующую криптограмму:

QRZ GXULQJ WKLV WLPH VKDKUDCDG KDG ERUQH NLQJ VKDKULBDU WKUHH VRQV RQ WKH WKRXVDQG DQG ILUVW QLJKW ZKHQ VKH KDG HQGHG WKH WDOH RI PDDUXI VKH URVH DQG NLVVHG WKH JURXQG EHIRUH KLP VDBLQJ JUHDW NLQJ IRU D WKRXVDQG DQG RQH QLJKWV L KDYH EHHQ UHFRXQWLQJ WR BRX IDEOHV RI SDVW DJHV DQG OHJHQGV RI DQFLHQW NLQJV PDB L PDNH VR EROG DV WR FUDYH D IDYRU RI BRXU PDMHVWB HSLORJXH WDOHV IURP WKH WKRXVDQG DQG RQH QLJKWV

Известно, что текст зашифрован с помощью одноалфавитного подстановочного шифра. Поиск всех возможных ключей (взлом грубой силой) в данном случае практически невыполним.

При виде такого шифротекста любой криптоаналитик немедленно приступит к анализу частоты появления букв. Его результат приведен в следующей таблице:

Вхождени й символов D E F G H I J K L M N O P

 34 5 3 18 36 11 14 23 19 1 5 6 7

 10,27

% 1,51

% 0,91

% 5,44

% 10,88

% 3,32

% 4,23

% 6,95

% 5,74

% 0,30

% 1,51

% 1,81

% 2,11

Вхождени й символов Q R S T U V W X Y Z A B C

 31 27 2 0 18 26 23 10 3 2 0 1 6

 9,37% 8,16

% 0,60

% 0% 5,44% 7,85

% 6,95

% 3,02

% 0,91

% 0,60

% 0% 0,30

% 1,81

%

Так как зашифрованный текст сравнительно короткий, то к нему нельзя применить частотный анализ непосредственно, т.к. будет получена бессмыслица.

Начнем с того, что сконцентрируемся на трех буквах H, D и Q, которые в зашифрованном тексте встречаются чаще всех. Естественно предположить, что эти буквы представляют собой, по–видимому, наиболее часто встречающиеся буквы английского алфавита (но не обязательно в том же порядке).

Другими словами, можно предположить, что:

H = e | t | a D = e | t | a Q = e | t | a

Чтобы идентифицировать эти буквы, нужно применить частотный анализ более тонким образом. Вместо простого подсчета частоты появления букв, можно проанализировать, как часто они появляются рядом с другими буквами. Например, появляется ли буква H рядом со многими буквами, или же она стремится стоять только возле некоторых определенных? Если H – гласная буква, то она должна появляться около большинства других букв, если же H – согласная буква, то она будет избегать соседства с многими буквами.

Из таблицы соседства букв видно (см. файл Excel к лабораторной работе), что буквы H и D соседствуют с 19 другими буквами, а буква Q – только с 11, т.е. она значительно менее дружелюбна. Поэтому можно предположить, что буквы H и D – гласные, а буква Q – согласная.

Теперь зададимся вопросом, каким гласным соответствуют буквы H и D? Скорее всего, они представляют собой буквы e и a – две наиболее часто встречающиеся гласные в английском языке. Интересной особенностью является то, что сочетание HH появляется в шифрограмме 2 раза, а DD – 1 раз. Так как в английском тексте сочетание букв ee встречается значительно чаще, чем aa, то будем считать, что H = e, а D = a. Это предположение подтверждается тем, что в английском языке есть неопределенный артикль a, и нет однобуквенного слова e (в шифротексте однобуквенное слово D встречается 2 раза, а слово H – ни разу).

В шифротексте есть еще одно однобуквенное слово – L и это означает, что буква L

представляет собой второе однобуквенное слово английского языка i1.

Хотя у нас есть пробелы между словами, однако следующий прием сработает и там, где в шифротексте пробелы отсутствуют. Этот прием позволит нам определить букву h после того, как мы нашли букву e. В английском языке буква h часто стоит перед буквой e (например, в the, they, then и т.д.), но очень редко после e. Из таблиц файла Excel видно, что буква K встречается перед H семь раз, а после H – ни разу, поэтому разумно предположить, что K = h.

Сделав в шифротексте замену H = e, D = a, L = i, K = h, получим:

QRZ GXUiQJ WhiV WiPe VhahUaCaG haG ERUQe NiQJ VhahUiBaU WhUee VRQV RQ Whe WhRXVaQG aQG IiUVW QiJhW ZheQ Vhe haG eQGeG Whe WaOe RI PaaUXI Vhe URVe aQG NiVVeG Whe JURXQG EeIRUe hiP VaBiQJ JUeaW NiQJ IRU a WhRXVaQG aQG RQe QiJhWV i haYe EeeQ UeFRXQWiQJ WR BRX IaEOeV RI SaVW aJeV aQG OeJeQGV RI aQFieQW NiQJV PaB i PaNe VR EROG aV WR FUaYe a IaYRU RI BRXU PaMeVWB

eSiORJXe WaOeV IURP Whe WhRXVaQG aQG RQe QiJhWV

1 Поиск однобуквенных слов является стандартным криптоаналитическим приемом. Этот прием срабатывает только потому, что в шифротексте между словами остались пробелы. Но зачастую криптографы удаляют все пробелы, чтобы затруднить противнику дешифрование сообщения.

При замене мы придерживались правила: буквы зашифрованного текста остаются прописными, а подставляемые буквы для открытого текста делаются строчными. Такая замена дает возможность определить еще несколько букв, поскольку теперь мы можем угадать некоторые слова в шифротексте. К примеру, самыми часто встречающимися трехбуквенными словами являются the и and и их легко найти в тексте, откуда получим W = t, Q = n, G = d.

Сделав теперь в шифротексте замену W = t, Q = n, G = d, получим:

nRZ dXUinJ thiV tiPe VhahUaCad had ERUne NinJ VhahUiBaU thUee VRnV Rn the thRXVand and IiUVt niJht Zhen Vhe had ended the taOe RI PaaUXI Vhe URVe and NiVVed the JURXnd EeIRUe hiP VaBinJ JUeat NinJ IRU a thRXVand and Rne niJhtV i haYe Eeen UeFRXntinJ tR BRX IaEOeV RI SaVt aJeV and OeJendV RI anFient NinJV PaB i PaNe VR EROd aV tR FUaYe a IaYRU RI BRXU PaMeVtB eSiORJXe taOeV IURP the thRXVand and Rne niJhtV

Дальнейший процесс дешифрования продолжается очень быстро. Действительно, теперь нетрудно угадать буквы R = o, V = s, P = m и после замены R = o, V = s, P = m, получим:

noZ dXUinJ this time shahUaCad had EoUne NinJ shahUiBaU thUee sons on the thoXsand and IiUst niJht Zhen she had ended the taOe oI maaUXI she Uose and Nissed the JUoXnd EeIoUe him saBinJ JUeat NinJ IoU a thoXsand and one niJhts i haYe Eeen UeFoXntinJ to BoX IaEOes oI Sast aJes and OeJends oI anFient NinJs maB i maNe so EoOd as to FUaYe a IaYoU oI BoXU maMestB eSiOoJXe taOes IUom the thoXsand and one niJhts

Следующая замена O = l, X = u, J = g дает:

noZ duUing this time shahUaCad had EoUne Ning shahUiBaU thUee sons on the thousand and IiUst night Zhen she had ended the tale oI maaUuI she Uose and Nissed the gUound EeIoUe him saBing gUeat Ning IoU a thousand and one nights i haYe Eeen UeFounting to Bou IaEles oI Sast ages and legends oI anFient Nings maB i maNe so Eold as to FUaYe a IaYoU oI BouU maMestB eSilogue tales IUom the thousand and one nights

Следующая замена U = r, F = c, I = f дает:

noZ during this time shahraCad had Eorne Ning shahriBar three sons on the thousand and first night Zhen she had ended the tale of maaruf she rose and Nissed the ground Eefore him saBing great Ning for a thousand and one nights i haYe Eeen recounting to Bou faEles of Sast ages and legends of ancient Nings maB i maNe so Eold as to craYe a faYor of Bour maMestB eSilogue tales from the thousand and one nights

Следующая замена Y = v, M = j, B = y дает:

noZ during this time shahraCad had Eorne Ning shahriyar three sons on the thousand and first night Zhen she had ended the tale of maaruf she rose and Nissed the ground Eefore him saying great Ning for a thousand and one nights i have Eeen recounting to you faEles of Sast ages and legends of ancient Nings may i maNe so Eold as to crave a favor of your majesty eSilogue tales from the thousand and one nights

Следующая замена Z = w, S = p, N = k дает:

now during this time shahraCad had Eorne king shahriyar three sons on the thousand and first night when she had ended the tale of maaruf she rose and kissed the ground Eefore him saying great king

for a thousand and one nights i have Eeen recounting to you faEles of past ages and legends of ancient kings may i make so Eold as to crave a favor of your majesty epilogue tales from the thousand and one nights

Наконец, последняя замена E = b, C = z приводит к открытому тексту:

now during this time shahrazad had borne king shahriyar three sons on the thousand and first night when she had ended the tale of maaruf she rose and kissed the ground before him saying great king for a thousand and one nights i have been recounting to you fables of past ages and legends of ancient kings may i make so bold as to crave a favor of your majesty epilogue tales from the thousand and one nights или окончательно: Now during this time Shahrazad had borne king Shahriyar three sons. On the thousand and first night when she had ended the tale of Maaruf she rose and kissed the ground before him saying: “Great king! For a thousand and one nights I have been recounting to you fables of past ages and legends of ancient kings. May I make so bold as to crave a favor of your majesty?”

Epilogue, Tales from the thousand and one nights

В исходной криптограмме ни разу не встречаются две буквы шифроалфавита - T и A

(буквы q и x открытого текста), поэтому таблица подстановки может иметь вид:

Алфавит: 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

Шифралфа вит:

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

или

Алфавит: 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

Шифралфа вит:

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

Сравнивая эти таблицы, естественно предположить, что для шифрования открытого текста использовался шифр Цезаря с величиной сдвига L = 3.

Файл Excel, прилагаемый к данной лабораторной работе, включает четыре пользовательских функции поддержки частотного анализа криптограмм:

1. Функция SymCount(c,sym) подсчитывает число вхождений символа sym в шифротекст.

Входной параметр с функции - ячейка рабочего листа Excel, содержащая анализируемый шифротекст.

Входной параметр sym функции - ячейка рабочего листа Excel, содержащая символ

sym, для которого подсчитывается число вхождений в шифротекст. Программный код функции SymCount(c,sym) имеет вид:

Function SymCount(c As Range, sym As Range) As Long Dim cip As String ' - строка шифротекста

Dim cLen As Long ' - длина строки шифротекста

Dim i As Long ' - счетчик циклов

Dim cnt As Long ' - счетчик вхождений символа sym в шифротекст

cip = Trim(c) cLen = Len(cip) cnt = 0

For i = 1 To cLen

If sym = Mid(cip, i, 1) Then cnt = cnt + 1 Next

SymCount = cnt End Function

2. Функция NeighboursCount(c,sym1,sym2) подсчитывает суммарное число размещений символов sym1 и sym2 по соседству друг с другом в шифротексте c.

Входной параметр с функции - ячейка рабочего листа Excel, содержащая анализируемый шифротекст.

Входные параметры sym1 и sym2 функции - ячейки рабочего листа Excel,

содержащие символы, для которых подсчитывается число соседних размещений.

Программный код функции NeighboursCount(c,sym1,sym2) имеет вид:

Function NeighboursCount(c As Range, sym1 As Range, sym2 As Range) As Long

Dim cip As String ' - строка шифротекста

Dim cLen As Long ' - длина строки шифротекста

Dim bi1 As String ' - биграмма 1 Dim bi2 As String ' - биграмма 2 Dim i As Long ' - счетчик циклов

Dim cnt As Long ' - счетчик биграмм

cip = Trim(c) cLen = Len(cip)

bi1 = Trim(sym1) & Trim(sym2) bi2 = Trim(sym2) & Trim(sym1) cnt = 0

For i = 1 To cLen - 1

If bi1 = Mid(cip, i, 2) Then cnt = cnt + 1

If (bi2 = Mid(cip, i, 2)) And (bi1 <> bi2) Then cnt = cnt

+ 1

Next

NeighboursCount = cnt End Function

3. Функция NextNeighboursCount(c,sym1,sym2) подсчитывает, сколько раз в шифротексте за символом sym1 следует символ sym2.

Входной параметр с функции - ячейка рабочего листа Excel, содержащая анализируемый шифротекст.

Входные параметры sym1 и sym2 функции - ячейки рабочего листа Excel,

содержащие символы, для которых подсчитывается число соседних размещений.

Программный код функции NextNeighboursCount(c,sym1,sym2) имеет вид:

Function NextNeighboursCount(c As Range, sym1 As Range, sym2 As Range) As Long

Dim cip As String ' - строка шифротекста

Dim cLen As Long ' - длина строки шифротекста

Dim bi As String ' - биграмма 1 Dim i As Long ' - счетчик циклов

Dim cnt As Long ' - счетчик биграмм 1

cip = Trim(c) cLen = Len(cip)

bi = Trim(sym1) & Trim(sym2) cnt = 0

For i = 1 To cLen - 1

If bi = Mid(cip, i, 2) Then cnt = cnt + 1 Next

NextNeighboursCount = cnt End Function

4. Функция PrevNeighboursCount(c,sym1,sym2) подсчитывает, сколько раз в шифротексте символ sym2 предшествует символу sym1.

Входной параметр с функции - ячейка рабочего листа Excel, содержащая анализируемый шифротекст.

Входные параметры sym1 и sym2 функции - ячейки рабочего листа Excel,

содержащие символы, для которых подсчитывается число соседних размещений.

Программный код функции PrevNeighboursCount(c,sym1,sym2) имеет вид:

Function PrevNeighboursCount(c As Range, sym1 As Range, sym2 As Range) As Long

Dim cip As String ' - строка шифротекста

Dim cLen As Long ' - длина строки шифротекста

Dim bi As String ' - биграмма вида \*X Dim i As Long ' - счетчик циклов

Dim cnt As Long ' - счетчик биграмм вида \*X

cip = Trim(c) cLen = Len(cip)

bi = Trim(sym2) & Trim(sym1) cnt = 0

For i = 1 To cLen - 1

If bi = Mid(cip, i, 2) Then cnt = cnt + 1 Next

PrevNeighboursCount = cnt End Function

Задание для выполнения

1. Изучите пользовательские функции SymCount(c), NeighboursCount(c,sym1,sym2), NextNeighboursCount(c,sym1,sym2), PrevNeighboursCount(c,sym1,sym2) в файле Excel, прилагаемом к лабораторной работе, и научитесь пользоваться ими для криптографического анализа шифртекстов.

2. Выберите примерно одну страницу произвольного русского текста. В редакторе Word удалите из этого текста символы конца абзаца. Скопируйте и ставьте полученный текст в ячейку Excel.

3. Используя функции PlainRusTextA(m) и EncryptRusTextA(m,L) из лабораторной работы №1, зашифруйте выбранный русский текст шифром Цезаря со сдвигом L = 5.

4. Дешифруйте полученный в работе шифротекст методом криптоаналитического частотного анализа.

5. В отчете по лабораторной работе приведите ход ваших рассуждений при взломе шифра.

Контрольные вопросы

1. Каковы особенности криптоаналитического частотного анализа?

2. Какой результат может быть получен в результате криптоаналитического частотного анализа?

3. Что дает частотный анализ?

Практическое занятие № 14. Исследование работы простейших алгоритмов шифрования в среде программирования Delphi

Цель: закрепление навыков простейшего шифрования.

Литература 1, 2, 3, 4, 5, 6, 9, 11, 13, 14,15, 16, 17

Задание.

1.Реализовать шифр цезаря в среде программирования Delphi

Запустите инструментальную объектно-ориентированную среду программирования Delphi. (Пуск/Программы/Borland Delphi/Delphi)

Создайте новый проект и сохраните его в новой папке AtbashCezar.

Поместите на форму следующие компоненты со страниц: Standard:

Label 3 шт;

Memo 2 шт;

ComboBox 1 шт;

RadioGroup 1 шт;

CheckBox 1 шт;

ActionList 1 шт. Additional:

BitBtn 7 шт. Dialogs:

OpenDialog 1 шт;

SaveDialog 1 шт.

Задайте следующие свойства данным компонентам:

Компонент Свойство Значение

Form1 Caption Шифрование текста

Label1 Caption Выберите шифр

Label2 Caption Исходный текст

Label3 Caption Результат

ComboBox1 Items Шифр Цезаря

ComboBox1 ItemIndex 0

RadioGroup1 ItemIndex 0

RadioGroup1 Items Шифровать

Расшифровать

RadioGroup1 Caption Действие

CheckBox1 Name Clr

CheckBox1 Caption Всегда очищать результат

BitBtn1 Caption Выполнить

BitBtn2 Caption Очистить

BitBtn3 Caption Очистить

BitBtn4 Caption Поменять

BitBtn5 Caption Выход

BitBtn6 Caption Открыть исх. текст из файла

BitBtn7 Caption Сохранить результат в файл

SaveDialog1 Filter Текстовые файлы|\*.txt

OpenDialog1 Filter Текстовые файлы|\*.txt

ActionList1 Name Crypto

Свойства Glyph(иконка, файл \*.ico) компонента BitBtn заполните самостоятельно. Стандартные иконки для кнопок Delphi расположены в C:\Program files\Common files\Borland Shared\Images\Buttons. Также вы можете изменить свойство Font компонентов Label1 и Label2.

{Шифр Цезаря. Шифрование}

function cEcezar(s:string; key:byte):string; var i:integer; c,d:char;rez:string;

begin

rez:='';

for i:=1 to length(s) do begin

c:=s[i];

d:=chr((ord(c)+ key)mod 255); rez:=rez+d;

end; cEcezar:=rez; end;

{Шифр Цезаря. Расшифрование} function cDcezar(s:string;key:byte):string; var i:integer; c,d:char;rez:string;

begin rez:='';

for i:=1 to length(s) do begin

c:=s[i];

d:=chr((ord(c)- key)mod 255); rez:=rez+d;

end; cDcezar:=rez; end;

Двойным щелчком вызовите редактор компонента Crypto (TActionList) и создайте следующие действия:

Для переименования в соответствии с образцом измените свойство Name у каждого действия.

Для каждого из действий опишите подпрограммы:

procedure TForm1.AtbashExecute(Sender: TObject); var i:integer;

begin

for i:=0 to memo1.Lines.Count do memo2.Lines.Add(catbash(memo1.Lines[i])); end;

procedure TForm1.ECezarExecute(Sender: TObject); var i:integer;key:byte;

begin

key:=strtoint(inputbox('Шифр Цезаря','Введите значение ключа - только целое число','3')); for i:=0 to memo1.Lines.Count do

memo2.Lines.Add(cEcezar(memo1.Lines[i],key)); end;

procedure TForm1.DCezarExecute(Sender: TObject); var i:integer;key:byte;

begin

key:=strtoint(inputbox('Шифр Цезаря','Введите значение ключа - только целое число','3')); for i:=0 to memo1.Lines.Count do

memo2.Lines.Add(cDcezar(memo1.Lines[i],key)); end;

procedure TForm1.clearExecute(Sender: TObject); begin

memo2.Clear;

end;

procedure TForm1.changeExecute(Sender: TObject); begin

memo1.Lines:=memo2.Lines; memo2.Clear;

end;

Опишите подпрограммы для каждой из кнопок BitBtn:

procedure TForm1.BitBtn1Click(Sender: TObject); begin

if clr.Checked then memo2.clear;

if combobox1.ItemIndex=0 then atbash.Execute else if action.ItemIndex=0 then ECezar.Execute else DCezar.Execute;

end;

procedure TForm1.BitBtn2Click(Sender: TObject); begin

memo1.Clear; end;

procedure TForm1.BitBtn3Click(Sender: TObject); begin

memo2.Clear; end;

procedure TForm1.BitBtn4Click(Sender: TObject); begin

change.Execute; end;

procedure TForm1.BitBtn5Click(Sender: TObject); begin

close; end;

procedure TForm1.BitBtn6Click(Sender: TObject); begin

if Opendialog1.Execute then memo1.Lines.LoadFromFile(opendialog1.FileName); end;

procedure TForm1.BitBtn7Click(Sender: TObject); begin

if Savedialog1.Execute then memo2.Lines.SaveToFile(savedialog1.FileName); end;

Запустите проект.

Контрольные вопросы

1. Каковы особенности шифра Цезаря?

2. Как реализовать проект на Delphi?

3. Как организована работа с формами?

Практическое занятие № 15. Исследование функций гаммирования на языке программирования Java

Цель: Формирование навыков гаммирования.

Задание.

1. Изучить теорию.

2. Разобрать пример.

3. Выполнить гаммирование на примере текстового файла.

4. Повторить гаммирование на собственном примере.

5. Реализовать алгоритм на языке Java.

Литература 1, 2, 3, 6, 9, 11, 13, 14,15, 16, 17

Теоретические основы

Гаммирование. Суть этого метода состоит в том, что символы шифруемого текста последовательно складываются с символами некоторой специальной последовательности, которая называется гаммой. Иногда такой метод представляют как наложение гаммы на исходный текст, поэтому он получил название «гаммирование».Процедуру наложения гаммы на исходный текст можно осуществить двумя способами. При первом способе символы исходного текста и гаммы заменяются цифровыми эквивалентами, которые затем складываются по модулю k, где k – число символов в алфавите, т.е. Ri = ( Si + G ) mod (k –1),

где Ri, Si, G — символы соответственно зашифрованного, исходного текста и гаммы.

При втором методе символы исходного текста и гаммы представляются в виде двоичного кода, затем соответствующие разряды складываются по модулю 2.

Вместо сложения по модулю 2 при гаммировании можно использовать и другие логические операции, например преобразование по правил логической эквивалентности и неэквивалентности. Такая замена равносильна введению еще одного ключа, который является выбор правила формирования символов зашифрованного сообщения из символов исходного текста и гаммы (Рис 1).

Стойкость шифрования методом гаммирования определяется главным образом свойством гаммы – длительностью периода и равномерностью статистических характеристик. Последнее свойство обеспечивает отсутствие закономерностей в появлении различных символов в пределах периода.

Обычно разделяют две разновидности гаммирования – с конечной и бесконечной гаммами. При хороших статистических свойствах гаммы стойкость шифрования определяется только длинной периода гаммы. При этом, если длина периода гаммы превышает длину шифруемого текста, то такой шифр теоретически является абсолютно стойким, т.е. его нельзя вскрыть при помощи статистической обработки зашифрованного текста. Это, однако, не означает, что дешифрование такого текста вообще невозможно: при

наличии некоторой дополнительной информации исходный текст может быть частично или полностью восстановлен даже при использовании бесконечной гаммы.

В качестве гаммы может быть использована любая последовательность и т.п. При  случайных символов, например, последовательность цифр числа шифровании с помощью, например, аппаратного шифратора последовательность гаммы может формироваться с

помощью датчика псевдослучайных чисел (ПСЧ). настоящее время разработано несколько

алгоритмов работы таких характеристики гаммы.

датчиков, которые обеспечивают

Рассеивание

удовлетворительные

Рассеивание осуществляется по рекуррентной формуле

.

по рекуррентной

формуле

, . - соответственно

символ исходного текстового файла и символ рассеянного текстового файла.

Метод перестановки

Перемешивание

заключается в том, что символы

шифруемого текста

переставляются по

определенным правилам

внутри шифруемого блока

символов.

Рассмотрим некоторые разновидности этого метода, которые могут быть использованы в

автоматизированных системах. Самая простая перестановка — написать исходный текст

задом наперед и одновременно разбить шифрограмму на пятерки букв. Например, из фразы

ПУСТЬ

БУДЕТ

ТАК,

КАК

МЫ ХОТЕЛИ.

получится

такой

шифротекст: ИЛЕТО

ХЫМКА

ККАТТ

ЕДУБЪ

ТСУП

В последней группе (пятерке) не хватает одной

буквы. Значит, прежде чем

шифровать

исходное

выражение, следует

его дополнить незначащей буквой (например, О)

до числа,

кратного пяти:

ПУСТЬ-БУДЕТ-ТАККА-КМЫХО-ТЕЛИО.

Тогда шифрограмма, несмотря на столь незначительные изменения, другому:

ОИЛЕТ ОХЫМК АККАТ ТЕДУБ ЬТСУП

будет выглядеть по-

Кажется, ничего сложного, но при расшифровке проявляются серьезные неудобства. Во время Гражданской войны в США в ходу был такой шифр: исходную фразу писали в несколько строк. Например, по пятнадцать букв в каждой (с заполнением последней строки незначащими буквами).

П У С Т Ь Б У Д Е Т Т А К К А

К М Ы Х О Т Е Л И К Л М Н О П

После этого вертикальные столбцы по порядку писали в строку с разбивкой на пятерки букв:

ПКУМС ЫТХЬО БТУЕД ЛЕИТК ТЛАМК НКОАП

Если строки укоротить, а количество строк увеличить, то получится прямоугольник-решетка,

в который можно записывать

исходный текст. Но тут уже

потребуется предварительная

договоренность между адресатом и отправителем посланий, поскольку сама решетка может

быть различной длины-высоты, записывать в нее можно по строкам, по столбцам, по

спирали туда или по спирали обратно, можно писать и по диагоналями, а для шифрования

можно брать тоже различные направления. В общем, здесь масса вариантов.

Выполнение работы

1. Гаммирование

Содержание исходного текстового файла, длиной 250 байт:

Выхожу один я на дорогу;

Сквозь туман кремнистый путь блестит; Ночь тиха. Пустыня внемлет богу,

И звезда с звездою говорит.

В небесах торжественно и чудно! Спит земля в сиянье голубом...

Что же мне так больно и так трудно? Жду ль чего? жалею ли о чем?

Результат гаммирования с параметрами L = 2; g0 = 9:

Д"X/.;I…m—miІ< ’ќ¤ў»ъґЦЧмо9µ

3oBQO\_im< ‰›’ТЄ·їТяЙЬЯфюэbP0BP}XWmaёіk< џ©»¶СыЖЪЫлуцC #!:|y\*‹[\_kv|ЃК¤Ь¬°јЗНащ$р !"5zu ‡]^cp…}›ПЄЇє№БЦаЩеця D§V6;5GQЌ•Ovx‹В’™©±ОшГ ддыI§)/@7MTџЁ±c‡ЊЗ–ћв·БВбШл\*ф

\*$.i:{VM`џzЃЌ‡™Јэз–ЅХ Ящ&эr\+.BEg’fl„їџ–¦

Результат гаммирования с параметрами L = 7; g0 = 19:

Й§".9Y™zѓљІшк юW.K`qyњчпіЯк §=tYmyЂ жГЬдю <Plm·™°ВЯкµ3GPmЙБЃµСй

.,Ќ’T‹њ°МСц\*я(BTaЃВ–¶ѕб-4пZ4BXm}ЊЯГЯн(Ehќs‘< ·МЧфCHэnNYh< ћ Илъ¤\*I]`v‘¤ёэШ#

<P–Ёlќ©Жбт EyNџѓЌ·ёЪЦ$ъ(CDduКЭр¬Ъй.gFZeі< ™¶яУу'+?„\_ЄЏђцЫм"6љЋGxљЪёЬэю¤- ‘…^k‰–ВчХе0V@A[

2. Рассеивание

Результат рассеивания исходного текстового файла:

병銡沄饞蚹歴蝧慹□葤□ ᄤ【２탤 늒膓坤 譫呲‹□□ 쏞躪穼続□ 匳□ 취麾뎓鎥榃

□ □샒죠 뛒醣튲銣決玌□ ぎ□®ᄕ □엒钦꺀 폡□곇료誥鲪晼枆□ □□ᄱ 퓰꿁

룔쮫늶짒醮빿쓞辤牴綒蕥鍳澊驛뷂 禆螙卫恀罟□偬 ＜ □뿘 샍責饹玃□ □ 퓢

3. Перемешивание

Результат перемешивания с параметром m = 3:

? нВвеы мхслоиежятун ьбоеод гигуно, л яуИ б нозамв .ед.зо.дрЧаот госу ;жз евС екмзвндоеоз юьт агткоу вмбоаорнли ьткн.ро е Вми н нитесабткеы сйта рхпу удттноьор

?жб елЖседтсувт еилнтьн; о ч Неиог чочь?у дтжниаохл!ае .юС пПлиуитс тозы енчмяел мяв

Результат перемешивания с параметром m = 17:

?днтзнв? гзхжтесиВаоиеао Поь а т ы тм рЖул тлбмычхси;лдидсутоеонйуо яотутбурюге джзтН р. ыомж у пнуваово лнмаел,туо екч гВья.нси ат!оз ьсу . т Икь ддт и;нчв.кво Сиортя еенЧре зббпнюуинСбгетенчволи дхькеомомнеелетягнаевс?л номзьс оо. оа ежи

3. Выводы

в результате гаммирования и рассеивания символы исходного текстового файла не сохраняются;

в результате перемешивания исходные символы текста не изменяются, а просто переставляются в определенной последовательности;

при разных параметрах гаммирования преобразованные файлы задаются различными символами.

4. Реализовать алгоритм на Java

Контрольные вопросы

1. Что такое гаммирование?

2. Привести пример гаммы.

3. Привести примеры замены и смешивания.

4. Как реализовать алгоритм гаммирования на языке программирования Java?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 16. ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ БЛОЧНОГО ШИФРОВАНИЯ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVA**

Цель: закрепление теоретических знаний блочного шифрования на конкретном примере.

Задачи:

1. Изучить теорию.

2. Ответить на вопросы.

3. Выполнить задание в среде MS Excel.

4. Самостоятельно найти решение поставленной задачи на языке программирования

Java и реализовать алгоритм.

5. Оформить результаты работы

Литература 1, 2, 3, 4, 6, 9, 11, 13, 14,15, 16, 17

Основные теоретические понятия

По мнению Шеннона, в практических симметричных шифрах необходимо использовать два общих принципа: рассеивание и перемешивание.

Рассеивание представляет собой распространение влияния одного знака открытого текста на много знаков шифротекста, что позволяет скрыть статистические свойства открытого текста.

Перемешивание предполагает использование таких шифрующих преобразований, которые усложняют восстановление взаимосвязи статистических свойств открытого и шифрованного текстов. Однако шифр должен не только затруднять раскрытие, но и обеспечивать легкость шифрования и дешифрования при известном пользователю секретном ключе.

Распространенным способом достижения эффектов рассеивания и перемешивания является использование составного шифра, т.е. такого шифра, который может быть реализован в виде последовательности простых шифров, каждый, из которых вносит свой вклад в значительное суммарное рассеивание и перемешивание.

В составных шифрах в качестве простых шифров чаще всего используются простые перестановки и подстановки. При перестановке перемешивают символы открытого текста, причем конкретный вид перемешивания определяется секретным ключом. При подстановке каждый символ открытого текста заменяют другим символом из того же алфавита, а конкретный вид подстановки также определяется секретным ключом. Следует заметить, что в современном блочном шифре блоки открытого текста и шифротекста представляют собой двоичные последовательности, обычно длиной 64 бита.

При многократном чередовании простых перестановок и подстановок, управляемых достаточно длинным секретным ключом, можно получить очень стойкий шифр с хорошим рассеиванием и перемешиванием. Криптоалгоритмы DES и отечественный стандарт шифрования данных ГОСТ 28147-89 построены в полном соответствии с этой методологией. Стандарт шифрования данных DES (Data Encryption Standard) опубликован Национальным бюро стандартов США в 1977 г. В 1980 г. он был принят в качестве стандарта шифрования данных США для защиты от несанкционированного доступа к важной, но

несекретной информации в государственных и коммерческих организациях.

К достоинствам DES можно отнести простоту ключевой системы, высокую скорость аппаратной и программной реализации, достаточно высокую криптографическую стойкость алгоритма шифрования при заданной длине ключа.

Первоначально метод, лежащий в основе стандарта DES, был разработан фирмой IBM для своих целей и реализован в виде системы "Люцифер" с ключом длиной 128 бит, управлявшим состояниями блоков перестановки и подстановки.

Алгоритм DES также использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битовых блоков данных с помощью 64-битового ключа, в котором значащими являются 56 бит, остальные 8 бит - проверочные биты контроля на четность.

Обобщенная схема алгоритма шифрования DES показана на рис. 1.

При шифровании очередного блока открытого текста T его биты подвергаются начальной перестановке с помощью табличной функции IP (см. файл DES.xls, прилагаемый к практической работе). Согласно этой функции бит 58 входного блока данных становится битом 1, бит 50 - битом 2, бит 9 – битом 60 и т.д. Полученные после начальной перестановки биты Т0 разделяется на две половины: L0 – левые 32 бита, R0 - правые 32 бита.

Затем выполняется итеративный процесс шифрования, состоящий из 16 раундов

(циклов), где результат i-й итерации Тi (Тi = LiRi) описывается формулами:

Li = Ri-1

Ri = Li-1  f(Ri-1, ki) i= 1,2, ..., 16

Используемая здесь функция f называется функцией шифрования. Ее аргументами являются биты Ri-1, полученные на предыдущей итерации, и 48-битовый ключ ki, который является результатом преобразования заданного ключа шифрования k (функция шифрования f и алгоритм получения ключей ki из ключа k описаны ниже).

На последнем шаге итерации получают биты L16 и R16, которые объединяются в 64-

битовую последовательность L16R16.

По окончании шифрования осуществляется восстановление позиций битов с помощью функции обратной перестановки IP-1 (см. файл DES.xls).

Процесс дешифрования данных является инверсным по отношению к процессу шифрования. Все действия должны быть выполнены в обратном порядке. Это означает, что расшифровываемые данные сначала переставляются в соответствии с функцией IP-1, а затем над последовательностью битов R16L16 выполняются те же действия, что и в процессе шифрования, но в обратном порядке. Итеративный процесс дешифрования может быть описан следующими формулами: Ri-1 = Li Li-1 = Ri  f(Li, ki)

Рис. 1. Схема алгоритма DES

Таким образом, для процесса дешифрова ия с переставленным входным блоком

L16R16 на первой

итерации

используется ключ

k16, на второй итерации - k15, на 16-й

итерации

используется ключ k1. На последнем шаге

итерации будут

получены

последовательности биты L0 и R0, которые объединяются в 64-битовую последовательность

битов L0R0. Затем

эти биты

переставляются в соответствии с функцией IP. Результатом

такого преобразования является исходная последовательность битов (расшифрованный блок открытого текста).

Схема вычисления функции шифрования f(Ri-1, ki) показана на рис.2. Для вычисления значения функции f используются табличные функции: Е (расширение 32-х битов до 48);

S1, S2, ..., S8 (преобразование 6-битового числа в 4-битовое);

Р (перестановка 32-битовой последовательности битов).

Аргументами

функции шифрования f

являются

Ri-1 (32 бита)

и ключ

ki (48 бит). Функция расширения Е, выполняющая расширение 32 бит до 48, определена в файле DES.xls. В соответствии с определением первые три бита E(Ri-1) - это биты 32, 1 и 2, а последние - 31, 32, 1 (некоторые биты повторяются).

Полученный

результат

E(Ri-1)

складывается по модулю 2

(операция

XOR) со

значением ключа ki, затем разбивается на восемь 6-битовых блоков B1, В2, ..., B8.

Далее каждый из этих блоков используется как номер элемента в функциях Si, S2, …, S8, содержащих 4-битовые значения (файл DES.xls).

Рис. 2. Схема вычисления функции шифрования f.

Следует отметить, что

выбор элемента в

функции

Si осуществляется

достаточно

оригинальным

образом.

Пусть

на вход

функции

Si,

поступает

6-битовый блок Bi = b1 b2 b3 b4 b5 b6, тогда 2-битовое число b1 b6 указывает номер строки функции, а 4-битовое число b2 b3 b4 Ь5 - номер столбца. Например, если на вход функции Si поступает 6-битовый блок Bi = 100110, то 2-битовое число b1 b6 = 10(2) = 2(10) указывает строку с номером 2 функции Si, а 4-битовое число b2 b3 b4 b5 = 0011(2) = 3(10) указывает столбец с номером 3 функции Si. Это означает, что в функции Si блок Bi = 100110 выбирает элемент на пересечении строки 2 и столбца 3 (см. файл DES.xls), т.е. элемент 8(10) =1000(2).

Таким образом, совокупность 6-битовых блоков B1, В2,..., В8 обеспечивает выбор 4-

битового элемента в каждой из функций S1, S2,.... S8. В результате получаем 32-битовый

блок S1(B1) S2(B2)

S3(B3)...

S8(B8), который затем преобразуется с

помощью

функции

перестановки битов Р (файл DES.xls).

Нетрудно заметить, что на каждой итерации используется новое значение ключа ki длиной 48 бит. Значения ключей ki вычисляются из начального ключа шифрования k согласно алгоритму, показанному на рис.6.3.

Ключ k представляет

собой 64-битовый блок с 8 битами контроля по

четности,

расположенными в позициях 8, 16, 24, 32, 40, 48, 56, 64. Для удаления контрольных битов и

подготовки ключа

DES.xls).

к работе

используется функция G начальной обработки ключа (файл

Результат преобразования G(k) разбивается на две

половины

С0 и D0

по 28 бит

каждая. Первые четыре строки функции G определяют, как выбираются биты С0 (первым битом С0 будет бит 57 ключа шифра, затем бит 49 и т.д., а последними битами - биты 44 и 36 ключа).

Следующие четыре строки функции G определяют, как выбираются биты D0 (т.е. D0

будет состоять из битов 63, 55, 47, ..., 12, 4 ключа шифра).

Видно, что для генерации С0 и D0 не используются биты 8, 16, 24, 32, 40, 48, 56 и 64

ключа шифра. Эти

биты не

влияют на

шифрование и могут служить для других целей

(например, для контроля по является 56-битовым.

четности).

Таким образом, в действительности ключ шифра

После определения С0D0 рекурсивно вычисляются CiDi, i = 1, 2, …, 16. Для этого применяются операции циклического сдвига влево на один или два бита в зависимости от номера шага итерации согласно табличной функции сдвигов (файл DES.xls).

Рис. 3. Схема алгоритма вычисления ключей ki

Операции сдвига выполняются

для последовательностей Ci

и Di независимо.

Например, С3 получается посредством циклического сдвига С2 влево на две позиции, а D3 -

посредством сдвига

D2 влево на две

позиции,

С16 и D16 получаются из С15 и D15

посредством сдвига влево на одну позицию и т.д.

Ключ ki, определяемый на каждом шаге итерации, есть результат выбора конкретных

битов из

56- битовой последовательности СiDi и

их перестановки с

помощью

функции

конечной

обработки

ключа Н

(файл DES.xls). Как следует

из функции H, первым битом

ключа ki будет 14-й бит СiDi, вторым - 17-й бит, 47-м битом будет 29-й бит СiDi, а 48-м битом - 32-й бит СiDi.

Задание для выполнения

1. Изучите процедуру

шифрования блока открытого

текста посредством

алгоритма

DES, пользуясь прилагаемым к лабораторной работе файлом DES.xls.

2. Зашифруйте с помощью алгоритма DES собственный 8-байтовый блок открытого

текста.

3.

В файле

DES.xls создайте новый рабочий лист

Decrypt и

реализуйте на нем

табличную процедуру дешифрования полученной криптограммы с помощью алгоритма DES.

4. В отчете

опишите

шаги дешифрования криптограммы, использованные вами

функции обратного преобразования и формулы Excel, реализующие каждый шаг процедуры дешифрования.

Контрольные вопросы для самопроверки

1. В чем особенности алгоритма DES

2. В чем отличия DES от RSA

3. Как с помощью электронных таблиц можно реализовать табличную процедуру шифрования и дешифрования полученной криптограммы?

4. В чем сущность процесса шифрования открытого текста?

5. Каковы особенности блочного шифрования?

**Практическое занятие №17. Исследование возможностей поточного шифрования на языке программирования Java**

Цель: Ознакомится с применением криптографических хэш-функций для обеспечения целостности информации.

Задачи:

1. Усвоить основные понятия.

2. Ответить на вопросы.

3. Реализовать программу на языке Java, вычисляющую значение хэш-функции произвольной информации и проверяющую целостность информации.

4. Провести тест реализованной программы.

5. Оформить результаты выполнения практической работы в виде отчета.

Литература 1, 2, 3, 6, 9, 11, 13, 14,15, 16, 17

Основные понятия

Хеширование (hashing) — преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциями или функциями свёртки, а их результаты называют хешем, хеш- кодом или дайджестом сообщения (message digest).

Хеширование применяется для сравнения данных: если у двух массивов хеш-коды разные, массивы гарантированно различаются; если одинаковые — массивы, скорее всего, одинаковы. В общем случае однозначного соответствия между исходными данными и хеш- кодом нет в силу того, что количество значений хеш-функций меньше, чем вариантов входного массива; существует множество массивов, дающих одинаковые хеш-коды — так называемые коллизии. Вероятность возникновения коллизий играет немаловажную роль в оценке качества хеш-функций.

Применение:

1. Циклический избыточный код (в частности, CRC8, CRC16, CRC32)

применяется для проверки целостности передачи данных.

2. MD5 и другие криптографические хеш-функции используются для подтверждения целостности и подлинности передаваемых данных.

3. Под названием «контрольное число» входит в состав номеров товаров и различных документов.

Задание для самостоятельной работы

Реализовать программу, которая выполняет следующие функции:

1. Вычисляет хэш-функцию по заданному алгоритму (MD2, MD4, MD5, SHA1).

2. Проверяет целостность данных по эталонному значению хэш-функции. Протестировать программу следующим образом:

1. Указать неверное название алгоритма хэширования.

2. Вычислить хэш данных, затем изменить данные и выполнить проверку целостности данных.

3. Вычислить хэш данных, затем изменить хэш данных и выполнить проверку целостности данных.

Ход работы:

1. Запустить среду программирования NetBeans

2. Создать новый проект. File – New Project. Выбрать тип проекта, как показано ниже:

3. Задать название проекта, его расположение и название главного класса:

4. Разверните узел созданного проекта и щелкните правой кнопкой по Source Packages

и из меню New выберите Java Class:

5. Задайте имя класса и его расположение (package):

6. Внесите изменения в класс, чтобы он выглядел следующим образом:

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException; import java.util.logging.Level;

import java.util.logging.Logger; public class myHash {

//переменная для хранения алгоритма хэширования

private String mdAlgorithm;

//конструктор объекта хэширования

public myHash(String algorithm) { mdAlgorithm = algorithm;

}

//вычисление значения хэш-функции

public String hashThis(byte[] data) { MessageDigest md = null;

try {

md = MessageDigest.getInstance(mdAlgorithm);

} catch (NoSuchAlgorithmException ex) {

Logger.getLogger(myHash.class.getName()).log(Level.SEVERE, null, ex);

}

byte[] digest = md.digest(data); return HexBin.encode(digest);

}

//проверить целостность данных

public boolean testData(byte[] data, String hash) { String h = hashThis(data);

return hash.equals(h);

}

}

7. Откройте файл Main.java и в функцию main внесите следующие строки:

8. Самостоятельно сделайте вызов функции testData и выведите результат.

Контрольные вопросы

1. Предложите свои варианты применения хэш-функции.

2. В чем основное отличие криптографической хэш-функции и контрольной суммы.

**Практическое занятие № 18. Программная реализация шифров на языке программирования Java**

Цель: на основании готового программного продукта закрепить навыки реализации алгоритмов блочного шифровния на языке программирования Java.

Задание:

1. Рассмотреть пример программной реализации в Python

2. Выполнить аналог на Java.

3. Оформить результаты работы

Литература 1, 2, 3, 4, 6, 9, 11, 13, 14,15, 16, 17

Основные теоретические понятия

Python — это бесплатный язык программирования с открытым исходным кодом, разработанный Гвидо ван Россумом (Guido van Rossum) в 1989 г. Он сочетает в себе удобство в использовании со способностью работать на многих платформах, поскольку реализован с учетом особенностей всех основных операционных систем. С тех пор Python непрерывно развивался и превратился в один из мощных современных языков программирования.

Лицензия по использованию этого языка не предусматривает никаких ограничений. Его можно изменять, дополнять, передавать третьим лицам, продавать и даже беспрепятственно распространять для коммерческого использования. Лицензионным соглашением обусловлена только защита автора от судебных проблем, которые могут появиться, если кто-то захочет предъявить автору претензии в связи с убытками, возникшими в результате применения Python, или если кто-то другой попытается присвоить себе права собственности на этот язык.

Применительно к дисциплине «Криптографическая защита информации» язык Python

обладает двумя важными преимуществами:

• поддержкой целочисленной арифметики произвольной длины;

• наличием криптографического пакета расширения языка.

В настоящее время в развитие языка Python вносит вклад многочисленная армия разработчиков. Если вы уделите время для посещения официального web-сайта языка Python, то сможете получить список разработчиков, которые интенсивно работают над развитием поддержки в языке Python новых технологий, например, XML и обработки изображений. С этого же web-сайта можно скачать дистрибутив языка Python (в лабораторной работе используется версия Python 2.6.5/2.6.6) и установить его на свой компьютер.

Для выполнения лабораторной работы потребуется дистрибутив пакета расширения Python Cryptography Toolkit pycrypto-2.1.0.win32-py2.6, который можно найти в Интернете. Этот пакет содержит набор криптографических алгоритмов и протоколов, реализованных для использования в Python. Документацию PyCrypto API Documentation по криптографическим объектам этого пакета также можно найти в Интернете.

Режимы работы блочных алгоритмов шифрования

Блочные алгоритмы шифрования могут работать в нескольких режимах:

• электронная кодовая книга ЕСВ (Electronic Code Book);

• сцепление блоков шифра СВС (Cipher Block Chaining);

• обратная связь по шифротексту CFB (Cipher Feed Back);

• обратная связь по выходу OFB (Output Feed Back).

Режим EBC "Электронная кодовая книга"

В этом режиме длинный файл разбивают на блоки заданной длины, и каждый из блоков шифруют независимо на одном и том же ключе шифрования.

Основное достоинство этого режима работы алгоритма - простота реализации, недостаток - слабая устойчивость против квалифицированных криптоаналитиков, т.к. из-за фиксированного характера шифрования при ограниченной длине блока возможно проведение криптоанализа "со словарем".

Режим CBC "Сцепление блоков шифра"

В этом режиме исходный файл разбивается на блоки M1M2…Mn. Первый блок M1 складывается по модулю 2 с начальным вектором IV (Initialization Vector), который меняется ежедневно и держится в секрете. Полученная сумма затем шифруется с использованием ключа шифрования, известного отправителю и получателю информации. Полученный блок шифротекста C1 складывается по модулю 2 со вторым блоком открытого текста, результат шифруется и получается второй блок шифротекста C2 и т.д. Процедура повторяется до тех пор, пока не будут обработаны все блоки текста.

Таким образом, для всех i = 1… n, где n - число блоков, результат шифрования Ci

определяется следующим образом: Ci = E(MiCi-1), где C0 = IV - вектор инициализации.

Очевидно, что последний блок шифротекста является функцией секретного ключа, начального вектора и каждого бита открытого текста независимо от его длины. Этот блок шифротекста называют кодом аутентификации сообщения (КАС).

Код КАС может быть легко проверен получателем, владеющим секретным ключом и начальным вектором, путем повторения процедуры, выполненной отправителем. Посторонний, однако, не может осуществить генерацию КАС, который воспринялся бы получателем как подлинный, чтобы добавить его к ложному сообщению, либо отделить КАС от истинного сообщения для использования его с измененным или ложным сообщением.

Режим CFB "Обратная связь по шифротексту"

В этом режиме размер блока (рис.6.4) может отличаться от заданного шифром размера m. Файл, подлежащий шифрованию, считывается последовательными блоками длиной k битов (k = 1 ...m).

Входной блок (m-битовый регистр сдвига) вначале содержит вектор инициализации. Предположим, что в результате разбиения открытого текста на блоки мы получили l блоков длиной k битов каждый (остаток дописывается нулями или пробелами). Тогда для любого i

=1...l блок Ci шифротекста будет равен Ci = Mi  Pi-1, где Рi-1 обозначает k старших битов предыдущего зашифрованного блока.

Обновление сдвигового регистра осуществляется путем удаления его старших k битов и сохранения Сi.

Восстановление зашифрованных данных выполняется относительно просто: Рi-1 и Сi

вычисляются аналогичным образом и Мi = Сi  Рi-1.

Режим OFB "Обратная связь по выходу"

Этот режим также использует переменный размер блока и сдвиговый регистр, инициализируемый так же, как в режиме CFB, а именно: входной блок вначале содержит вектор инициализации IV. При этом для каждого сеанса шифрования данных необходимо использовать новое начальное состояние регистра, которое должно пересылаться по каналу открытым текстом.

Отличие от режима обратной связи по шифротексту состоит в методе обновления сдвигового регистра (рис.6.5).

Каждому из режимов ЕСВ, СВС, CFB, OFB свойственны свои достоинства и недостатки, что обусловливает области их применения.

Режим ЕСВ хорошо подходит для шифрования ключей, режим CFB, как правило, предназначается для шифрования отдельных символов, а режим OFB нередко применяется для шифрования в спутниковых системах связи. Режимы СВС и CFB также пригодны для аутентификации данных.

Первоначально блочные алгоритмы шифрования предназначались только для шифрования и дешифрования данных ЭВМ, однако впоследствии их применение было обобщено и на аутентификацию.

Суть проблемы аутентификации данных заключается в следующем. В системах автоматической обработки данных человек не в состоянии просмотреть данные, чтобы установить, не внесены ли в них какие-либо изменения. При огромных объемах данных, проходящих в современных системах обработки, просмотр занял бы слишком много времени. К тому же избыточность данных может оказаться недостаточной для обнаружения ошибок. Даже в тех случаях, когда просмотр человеком возможен, данные могут быть изменены так, что обнаружить эти изменения человеку трудно. Например, "do" может быть заменено на "do not", "$1900" - на "$9100".

Поэтому желательно иметь автоматическое средство обнаружения преднамеренных и непреднамеренных изменений данных. Обыкновенные коды, обнаруживающие ошибки, непригодны, так как если алгоритм образования кода известен, противник может выработать правильный код после внесения изменений в данные. Однако с помощью блочного алгоритма шифрования можно образовать криптографическую контрольную сумму (ККС), которая сможет защитить как от случайных, так и преднамеренных, но несанкционированных изменений данных.

Этот процесс описывает стандарт для аутентификации данных ЭВМ (FIPS 113). Суть стандарта состоит в том, что данные зашифровываются в режиме обратной связи по шифротексту (режим CFB) или в режиме сцепления блоков шифра (режим СВС), в результате получается окончательный блок шифра, представляющий собой функцию всех разрядов открытого текста. После этого сообщение, которое содержит открытый текст, может быть передано вместе с вычисленным окончательным блоком шифра, служащим в качестве ККС.

Одни и те же данные можно защитить, пользуясь как шифрованием, так и аутентификацией. Данные защищаются от ознакомления шифрованием, а изменения обнаруживаются посредством аутентификации. Алгоритм аутентификации можно применить как к открытому, так и к зашифрованному тексту.

Шифрование и аутентификацию используют для защиты данных, хранящихся в ЭВМ. Во многих ЭВМ пароли зашифровывают необратимым образом и хранят в памяти машины. Когда пользователь обращается к ЭВМ и вводит пароль, последний зашифровывается и сравнивается с хранящимся значением. Если обе зашифрованные величины одинаковы, пользователь получает доступ к машине, в противном случае следует отказ. Нередко зашифрованный пароль вырабатывают с помощью блочного алгоритма, причем ключ полагается равным паролю, а открытый текст - коду идентификации пользователя.

Одним из наиболее важных применений блочных алгоритмов является защита сообщений электронной системы платежей (ЭСП) при операциях с клиентами и между банками. Блочные алгоритмы шифрования используются в банковских автоматах, терминалах в торговых точках, АРМ и главных ЭВМ. Диапазон защищаемых ими данных весьма широк - от оплат $50 до переводов на многие миллионы долларов.

Реализация блочных алгоритмов шифрования в Python

В пакете расширения Python Cryptography Toolkit реализована поддержка следующих блочных алгоритмов шифрования: DES, DES3, ARC2, CAST, Blowfish, AES.

Параметры метода cryptoobject.new(key, mode, iv) блочных криптоалгоритмов

Алгоритм Длина блока, байт Длина ключа, байт Длина iv, байт Режимы работы алгоритма

DES кратная 8 8 8 CBC, CFB, ECB, OFB, PGP

DES3 кратная 8 16 | 24 8 CBC, CFB, ECB, OFB, PGP

ARC2 кратная 8 ≥1 8 CBC, CFB, ECB, OFB, PGP

CAST кратная 8 [5..16] 8 CBC, CFB, ECB, OFB, PGP

Blowfish кратная 8 ≥1 8 CBC, CFB, ECB, OFB, PGP

AES кратная 16 16 | 24 | 32 16 CBC, CFB, ECB, OFB, PGP

Программный код, шифрующий и дешифрующий несколько блоков открытого текста, для всех блочных алгоритмов шифрования имеет одинаковый вид, где имя объекта шифрования совпадает с названием алгоритма:

# -\*- coding: cp1251 -\*-

from Crypto.Cipher import DES # - импорт модуля DES

pwd='YourKeys' # ключ

iv8='01234567' # 8-байтовый вектор инициализации txt = 'Algoritm' # блок открытого текста print 'Открытый текст:', txt

# --- Шифрование открытого текста --- crpt=DES.new(pwd,DES.MODE\_ECB,iv8) # создать объект crpt # допустимые режимы работы шифра: ECB,CBC,CFB,OFB

c = crpt.encrypt(txt) print 'Шифротекст:', c

# --- Дешифрование криптограммы --- crpt=DES.new(pwd,DES.MODE\_ECB,iv8) # re\_init d =crpt.decrypt(c)

print 'Дешифрованный текст:', d

Ниже приведен листинг программы Python, которая выполняет шифрование текстового файла C:\Inetpub\data.txt в режиме CBC с помощью алгоритма AES на ключе abcdefghijklmnop и сохраняет полученный шифротекст в файле C:\Inetpub\edata.txt:

# -\*- coding: cp1251 -\*- from Crypto.Cipher import AES

N = 16

ss=' '

pwd='abcdefghijklmnop' iv16='0123456789ABCDEF'

# Считывание открытого текста из входного файла

fl = open('C:/Inetpub/data.txt','rb') s = fl.read()

fl.close

# Дополнение открытого текста пробелами до длины, кратной N lenght = len(s)

r=divmod(lenght, N) s = s + ss[0:N-r[1]]

# Шифрование открытого текста в режиме CBC print

print 'ОТКРЫТЫЙ ТЕКСТ:', s

cdo=AES.new(pwd,AES.MODE\_CBC,iv16) c = cdo.encrypt(s)

print

print 'ШИФРОТЕКСТ:', c

# Сохранение шифротекста в выходном файле fl = open('C:/Inetpub/edata.txt','wb') fl.write(c)

fl.close

Задание для выполнения

1. По согласованию с преподавателем выберите один из блочных алгоритмов шифрования, на языке программирования Python разработайте программу шифрования блока открытого текста выбранным алгоритмом шифрования и сохраните ее в файле с расширением \*.py.

2. С помощью созданной программы шифрования зашифруйте несколько блоков открытого текста в различных режимах работы алгоритма и на разных ключах шифрования.

3. Изучите листинг приведенной выше программы шифрования файла открытого текста криптографического шифра AES.

4. Создайте в редакторе Блокнот файл C:\Inetpub\data.txt с произвольным открытым текстом. Зашифруйте открытый текст с помощью шифра AES в режиме CFB, полученный шифротекст сохраните в файле C:\Inetpub\edata.txt.

5. Из полученного шифротекста выделите последний блок и используйте его в качестве контрольной суммы для проверки целостности текстового файла.

6. Разработайте программу дешифрования файла открытого текста с помощью алгоритма шифрования AES и дешифруйте с ее помощью файл C:\Inetpub\edata.txt.

7. Напишите программу на языке Python для шифрования файла открытого текста алгоритмом DES в режиме EBC целиком (аналогично шифрованию файла алгоритмом AES), а также по блокам в цикле while. Убедитесь, что шифрованные тексты, полученные этими двумя способами зашифровывания, одинаковы.

8. Напишите аналогичную программу на языке Java.

Контрольные вопросы

1. В чем особенности блочного шифрования?

2. Приведите примеры блочных шифров?

3. какой язык программирования легче для преализации блочного шифрования и какой целесообразнее использовать в настоящее время?

**Практическое занятие № 19. Исследование блочного алгоритма шифрования DES**

**в MS Excel**

Цель: закрепление теоретических знаний блочного шифрования на конкретном примере.

Задачи:

1. Изучить теорию.

2. Ответить на вопросы.

3. Выполнить задание

4. Оформить результаты работы

Литература 1, 2, 3, 4, 6, 9, 11, 13, 14,15, 16, 17

Основные понятия

По мнению Шеннона, в практических симметричных шифрах необходимо использовать два общих принципа: рассеивание и перемешивание.

Рассеивание представляет собой распространение влияния одного знака открытого текста на много знаков шифротекста, что позволяет скрыть статистические свойства открытого текста.

Перемешивание предполагает использование таких шифрующих преобразований, которые усложняют восстановление взаимосвязи статистических свойств открытого и шифрованного текстов. Однако шифр должен не только затруднять раскрытие, но и обеспечивать легкость шифрования и дешифрования при известном пользователю секретном ключе.

Распространенным способом достижения эффектов рассеивания и перемешивания является использование составного шифра, т.е. такого шифра, который может быть реализован в виде последовательности простых шифров, каждый, из которых вносит свой вклад в значительное суммарное рассеивание и перемешивание.

В составных шифрах в качестве простых шифров чаще всего используются простые перестановки и подстановки. При перестановке перемешивают символы открытого текста, причем конкретный вид перемешивания определяется секретным ключом. При подстановке каждый символ открытого текста заменяют другим символом из того же алфавита, а конкретный вид подстановки также определяется секретным ключом. Следует заметить, что в современном блочном шифре блоки открытого текста и шифротекста представляют собой двоичные последовательности, обычно длиной 64 бита.

При многократном чередовании простых перестановок и подстановок, управляемых достаточно длинным секретным ключом, можно получить очень стойкий шифр с хорошим рассеиванием и перемешиванием. Криптоалгоритмы DES и отечественный стандарт шифрования данных ГОСТ 28147-89 построены в полном соответствии с этой методологией. Стандарт шифрования данных DES (Data Encryption Standard) опубликован Национальным бюро стандартов США в 1977 г. В 1980 г. он был принят в качестве стандарта шифрования данных США для защиты от несанкционированного доступа к важной, но

несекретной информации в государственных и коммерческих организациях.

К достоинствам DES можно отнести простоту ключевой системы, высокую скорость аппаратной и программной реализации, достаточно высокую криптографическую стойкость алгоритма шифрования при заданной длине ключа.

Первоначально метод, лежащий в основе стандарта DES, был разработан фирмой IBM для своих целей и реализован в виде системы "Люцифер" с ключом длиной 128 бит, управлявшим состояниями блоков перестановки и подстановки.

Алгоритм DES также использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битовых блоков данных с помощью 64-битового ключа, в котором значащими являются 56 бит, остальные 8 бит - проверочные биты контроля на четность.

Обобщенная схема алгоритма шифрования DES показана на рис. 1.

При шифровании очередного блока открытого текста T его биты подвергаются начальной перестановке с помощью табличной функции IP (см. файл DES.xls, прилагаемый к практической работе). Согласно этой функции бит 58 входного блока данных становится битом 1, бит 50 - битом 2, бит 9 – битом 60 и т.д. Полученные после начальной перестановки биты Т0 разделяется на две половины: L0 – левые 32 бита, R0 - правые 32 бита.

Затем выполняется итеративный процесс шифрования, состоящий из 16 раундов

(циклов), где результат i-й итерации Тi (Тi = LiRi) описывается формулами:

Li = Ri-1

Ri = Li-1  f(Ri-1, ki) i= 1,2, ..., 16

Используемая здесь функция f называется функцией шифрования. Ее аргументами являются биты Ri-1, полученные на предыдущей итерации, и 48-битовый ключ ki, который является результатом преобразования заданного ключа шифрования k (функция шифрования f и алгоритм получения ключей ki из ключа k описаны ниже).

На последнем шаге итерации получают биты L16 и R16, которые объединяются в 64-

битовую последовательность L16R16.

По окончании шифрования осуществляется восстановление позиций битов с помощью функции обратной перестановки IP-1 (см. файл DES.xls).

Процесс дешифрования данных является инверсным по отношению к процессу шифрования. Все действия должны быть выполнены в обратном порядке. Это означает, что расшифровываемые данные сначала переставляются в соответствии с функцией IP-1, а затем над последовательностью битов R16L16 выполняются те же действия, что и в процессе шифрования, но в обратном порядке.

Итеративный процесс дешифрования может быть описан следующими формулами: Ri-1 = Li

Li-1 = Ri  f(Li, ki)

Рис. 1. Схема алгоритма DES.

Таким образом, для процесса дешифрова ия с переставленным входным блоком

L16R16 на первой

итерации

используется ключ

k16, на второй итерации - k15, на 16-й

итерации

используется ключ k1. На последнем шаге

итерации будут

получены

последовательности биты L0 и R0, которые объединяются в 64-битовую последовательность

битов L0R0. Затем

эти биты

переставляются в соответствии с функцией IP. Результатом

такого преобразования является исходная последовательность битов (расшифрованный блок открытого текста).

Схема вычисления функции шифрования f(Ri-1, ki) показана на рис.2. Для вычисления значения функции f используются табличные функции: Е (расширение 32-х битов до 48);

S1, S2, ..., S8 (преобразование 6-битового числа в 4-битовое);

Р (перестановка 32-битовой последовательности битов).

Аргументами

функции шифрования f

являются

Ri-1 (32 бита)

и ключ

ki (48 бит). Функция расширения Е, выполняющая расширение 32 бит до 48, определена в файле DES.xls. В соответствии с определением первые три бита E(Ri-1) - это биты 32, 1 и 2, а последние - 31, 32, 1 (некоторые биты повторяются).

Полученный

результат

E(Ri-1)

складывается по модулю 2

(операция

XOR) со

значением ключа ki, затем разбивается на восемь 6-битовых блоков B1, В2, ..., B8.

Далее каждый из этих блоков используется как номер элемента в функциях Si, S2, …, S8, содержащих 4-битовые значения (файл DES.xls).

Рис. 2. Схема вычисления функции шифрования f.

Следует отметить, что

выбор элемента в

функции

Si осуществляется

достаточно

оригинальным

образом.

Пусть

на вход

функции

Si,

поступает

6-битовый блок Bi = b1 b2 b3 b4 b5 b6, тогда 2-битовое число b1 b6 указывает номер строки функции, а 4-битовое число b2 b3 b4 Ь5 - номер столбца. Например, если на вход функции Si поступает 6-битовый блок Bi = 100110, то 2-битовое число b1 b6 = 10(2) = 2(10) указывает строку с номером 2 функции Si, а 4-битовое число b2 b3 b4 b5 = 0011(2) = 3(10) указывает столбец с номером 3 функции Si. Это означает, что в функции Si блок Bi = 100110 выбирает элемент на пересечении строки 2 и столбца 3 (см. файл DES.xls), т.е. элемент 8(10) =1000(2).

Таким образом, совокупность 6-битовых блоков B1, В2,..., В8 обеспечивает выбор 4-

битового элемента в каждой из функций S1, S2,.... S8. В результате получаем 32-битовый

блок S1(B1) S2(B2)

S3(B3)...

S8(B8), который затем преобразуется с

помощью

функции

перестановки битов Р (файл DES.xls).

Нетрудно заметить, что на каждой итерации используется новое значение ключа ki длиной 48 бит. Значения ключей ki вычисляются из начального ключа шифрования k согласно алгоритму, показанному на рис.6.3.

Ключ k представляет

собой 64-битовый блок с 8 битами контроля по

четности,

расположенными в позициях 8, 16, 24, 32, 40, 48, 56, 64. Для удаления контрольных битов и

подготовки ключа

DES.xls).

к работе

используется функция G начальной обработки ключа (файл

Результат преобразования G(k) разбивается на две

половины

С0 и D0

по 28 бит

каждая. Первые четыре строки функции G определяют, как выбираются биты С0 (первым битом С0 будет бит 57 ключа шифра, затем бит 49 и т.д., а последними битами - биты 44 и 36 ключа).

Следующие четыре строки функции G определяют, как выбираются биты D0 (т.е. D0

будет состоять из битов 63, 55, 47, ..., 12, 4 ключа шифра).

Видно, что для генерации С0 и D0 не используются биты 8, 16, 24, 32, 40, 48, 56 и 64

ключа шифра. Эти

биты не

влияют на

шифрование и могут служить для других целей

(например, для контроля по является 56-битовым.

четности).

Таким образом, в действительности ключ шифра

После определения С0D0 рекурсивно вычисляются CiDi, i = 1, 2, …, 16. Для этого применяются операции циклического сдвига влево на один или два бита в зависимости от номера шага итерации согласно табличной функции сдвигов (файл DES.xls).

Рис. 3. Схема алгоритма вычисления ключей ki

Операции сдвига выполняются

для последовательностей Ci

и Di независимо.

Например, С3 получается посредством циклического сдвига С2 влево на две позиции, а D3 -

посредством сдвига

D2 влево на две

позиции,

С16 и D16 получаются из С15 и D15

посредством сдвига влево на одну позицию и т.д.

Ключ ki, определяемый на каждом шаге итерации, есть результат выбора конкретных

битов из

56- битовой последовательности СiDi и

их перестановки с

помощью

функции

конечной

обработки

ключа Н

(файл DES.xls). Как следует

из функции H, первым битом

ключа ki будет 14-й бит СiDi, вторым - 17-й бит, 47-м битом будет 29-й бит СiDi, а 48-м битом - 32-й бит СiDi.

Задание для выполнения

1. Изучите процедуру

шифрования блока открытого

текста посредством

алгоритма

DES, пользуясь прилагаемым к лабораторной работе файлом DES.xls.

2. Зашифруйте с помощью алгоритма DES собственный 8-байтовый блок открытого

текста.

3.

В файле

DES.xls создайте новый рабочий лист

Decrypt и

реализуйте на нем

табличную процедуру дешифрования полученной криптограммы с помощью алгоритма DES.

4. В отчете

опишите

шаги дешифрования криптограммы, использованные вами

функции обратного преобразования и формулы Excel, реализующие каждый шаг процедуры дешифрования.

Контрольные вопросы для самопроверки

1. В чем особенности алгоритма DES

2. В чем отличия DES от RSA

3. Как с помощью электронных

таблиц можно реализовать табличную

процедуру

шифрования и дешифрования полученной криптограммы?

4. В чем сущность процесса шифрования открытого текста?

5. Каковы особенности блочного шифрования?

**Практическое занятие № 20. Исследование алгоритма шифрования RSA в MS**

Цель: Изучение методологии формирования открытого и закрытого ключей на сонове

RSA. Формирование навыков применения алгоритма.

Литература 2, 3, 6, 9, 11, 13, 14,15, 16, 17

Теоретические основы

Алгоритм создания открытого и секретного ключей.

Выбираются два случайных простых числа и каждое).

Вычисляется их произведение Вычисляется значение функции Эйлера от числа n:

заданного

размера (например, 512 битов

Выбирается целое

число ,

взаимно

простое со

значением функции .

Обычно в

качестве

e берут простые числа, содержащие небольшое количество

единичных битов в

двоичной записи, например, простые числа Ферма 17, 257, или 65537.

Вычисляется число d, мультипликативно обратное к числу удовлетворяющее сравнению:

e по модулю , т.е число,

2…).

; то есть

 , где k любое натуральное число (0, 1,

Пара Пара

публикуется в качестве открытого ключа RSA (RSA public key).

играет роль секретного ключа RSA (RSA secret key) и держится в секрете.

Число

называется

модулем, а числа

и — открытой и секретной экспонентами,

соответственно.

Шифрование и расшифрование Схема RSA

Предположим, сторона хочет послать стороне сообще

Сообщением являются целые числа лежащие от до

Алгоритм:

Взять открытый ключ стороны

Алгоритм:

Принять зашифрованное сообщение

Взять открытый текст

Применить

свой секретный ключ для

Передать шифрованное сообщение: расшифровки сообщения:

Корректность схемы RSA

Уравнения и , на которых основана схема RSA, определяют взаимно обратные преобразования множества

Доказательство Действительно, для

Докажем, что .

Возможны два случая:

 .

Поскольку числа и являются взаимно обратными относительно умножения по модулю

, т.e

для некоторого целого , тогда

где второе тождество следует из теоремы Ферма.

Таким образом, из Китайской теоремы об остатках Пример

 текст

 вычислить шифротекст

Расшифровани е вычислить исходное сообщение

Задание

1. Выполнить пример на основе собственных данных

2. Оформить работу и сдать

**Практическое занятие № 10.Исследование процесса формирования ЭЦП на основе RSA**

Цель: Ознакомится с применением криптографических хэш-функций для обеспечения целостности информации и формирования ЭЦП.

Задачи:

6. Усвоить основные понятия.

7. Ответить на вопросы.

8. Реализовать программу на языке Java, вычисляющую значение хэш-функции произвольной информации и проверяющую целостность информации.

9. Провести тест реализованной программы.

10. Оформить результаты выполнения практической работы в виде отчета.

6. Пояснить возможность применения хэш функции для формирования ЭЦП

7. Самостоятельно реализовать алгоритм формирования ЭЦП.

Литература 1, 2, 3, 4, 6, 9, 11, 13, 14,15, 16, 17

Основные понятия

Хеширование (hashing) — преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциями или функциями свёртки, а их результаты называют хешем, хеш- кодом или дайджестом сообщения (message digest).

Хеширование применяется для сравнения данных: если у двух массивов хеш-коды разные, массивы гарантированно различаются; если одинаковые — массивы, скорее всего, одинаковы. В общем случае однозначного соответствия между исходными данными и хеш- кодом нет в силу того, что количество значений хеш-функций меньше, чем вариантов входного массива; существует множество массивов, дающих одинаковые хеш-коды — так называемые коллизии. Вероятность возникновения коллизий играет немаловажную роль в оценке качества хеш-функций.

Применение:

4. Циклический избыточный код (в частности, CRC8, CRC16, CRC32) применяется для проверки целостности передачи данных.

5. MD5 и другие криптографические хеш-функции используются для подтверждения целостности и подлинности передаваемых данных.

6. Под названием «контрольное число» входит в состав номеров товаров и различных документов.

Вопросы:

1. Предложите свои варианты применения хэш-функции.

2. В чем основное отличие криптографической хэш-функции и контрольной суммы.

3. Как можно использовать эту функцию для формирования ЭЦП?

Задание для самостоятельной работы

Реализовать программу, которая выполняет следующие функции:

3. Вычисляет хэш-функцию по заданному алгоритму (MD2, MD4, MD5, SHA1).

4. Проверяет целостность данных по эталонному значению хэш-функции. Протестировать программу следующим образом:

4. Указать неверное название алгоритма хэширования.

5. Вычислить хэш данных, затем изменить данные и выполнить проверку целостности данных.

6. Вычислить хэш данных, затем изменить хэш данных и выполнить проверку целостности данных.

7. Реализовать хэш для формирования ЭЦП.

Ход работы:

1. Запустить среду программирования NetBeans

2. Создать новый проект. File – New Project. Выбрать тип проекта, как показано ниже:

3. Задать название проекта, его расположение и название главного класса:

4. Разверните узел созданного проекта и щелкните правой кнопкой по Source Packages

и из меню New выберите Java Class:

5. Задайте имя класса и его расположение (package):

6. Внесите изменения в класс, чтобы он выглядел следующим образом:

7. Откройте файл Main.java и в функцию main внесите следующие строки:

8. Самостоятельно сделайте вызов функции testData и выведите результат.

Контрольные вопросы для самопроверки

1. Что такое электронно-цифровая подпись?

2. Как формируется ЭЦП?

3. Для чего нужна функция хэш?

4. Для чего нужны алгоритмы MD2, MD4, MD5, SHA1

5. как проверить целостность?

**Практическое занятие № 21.**

**Программная реализация возможностей формирования ЭЦП на основе RSA на основе языка программирования Java**

Цель: освоение навыков программной реализации RSA.

Задание.

1. Ввести текст программы.

2. Проработать ее выполнение.

3. Доработать программу для ЭЦП

Литература 1, 2, 3, 4, 6, 9, 11, 13, 14,15, 16, 17

Среда разработки: Visual studio.Net 10.0

Теоретические основы

Текст программы. Проанализировать и ввести. package cryptor;

import java.math.BigInteger; import java.util.Random;

/\*\*

\*

\* @author Администратор

\*/

public class myRSA {

private BigInteger publicKey; private BigInteger secretKey; private BigInteger initN;

public void generateKeys() { Random r = new Random();

BigInteger p = BigInteger.probablePrime(16, r); r.nextInt();

BigInteger q = BigInteger.probablePrime(16, r); BigInteger n = p;

n = n.multiply(q);

//System.out.println("n = " + n);

BigInteger fn = p.subtract(BigInteger.ONE); fn = fn.multiply(q.subtract(BigInteger.ONE));

//System.out.println("fn = " + fn); BigInteger ee = BigInteger.ZERO; Long tmpe = (long) 2;

do {

tmpe++;

ee = fn.gcd(new BigInteger(tmpe.toString()));

} while (ee.compareTo(BigInteger.ONE) != 0); BigInteger e = new BigInteger(tmpe.toString());

//System.out.println("e = " + e); BigInteger d = e.modInverse(fn); publicKey = e;

secretKey = d; initN = n;

}

public BigInteger getPublicKey() { return publicKey;

}

public BigInteger getSecretKey() { return secretKey;

}

public BigInteger getInitN() { return initN;

}

public void setKeysN(BigInteger pub, BigInteger sec, BigInteger initN) { publicKey = pub;

secretKey = sec; this.initN = initN;

}

private BigInteger cryptBlock(byte[] block) { BigInteger result = new BigInteger(block); return result.modPow(publicKey, initN);

}

private byte[] decryptBlock(BigInteger block) {

return block.modPow(secretKey, initN).toByteArray();

}

public String[] cryptMessage(byte[] mes) { int d = 1;

//поблочное шифрование???

// for (int i = 1; i < 5; i++) {

// if (mes.length % i == 0) {

// d = i;

// }

// }

BigInteger[] res = new BigInteger[mes.length / d]; int k = 0;

byte[] tmp = new byte[d];

for (int i = 0; i < mes.length; i += d) { tmp = new byte[d];

for (int j = i; j < i + d; j++) { tmp[j - i] = mes[j];

}

res[k] = cryptBlock(tmp); k++;

}

//BigInteger pr = new BigInteger(tmp); String[] result = new String[res.length]; for (int i = 0; i < res.length; i++) {

result[i] = res[i].toString();

}

return result;

}

public byte[] decryptMessage(String[] crypt) { BigInteger[] cryptmes = new BigInteger[crypt.length]; for (int i=0; i<crypt.length; i++){

cryptmes[i] = new BigInteger(crypt[i]);

}

if (cryptmes.length == 0) { return null;

}

byte[] tmp = decryptBlock(cryptmes[0]); int d = tmp.length;

byte[] result = new byte[(int) (cryptmes.length \* d)]; for (int i = 0; i < d; i++) {

result[i] = tmp[i];

}

for (int i = 1; i < cryptmes.length; i++) {

tmp = decryptBlock(cryptmes[i]); for (int j = 0; j < tmp.length; j++) {

result[d] = tmp[j]; d++;

}

}

return result;

}

}

**Практическое занятие №22 Средства обеспечения безопасности ОС семейства Windows**

Цель: изучить модель безопасности операционной системы Windows, получить навыки практического использования ее средств обеспечения безопасности.

1. Основные сведения

1.1. Классификация защиты семейства ОС Windows

Разработчики операционной системы Windows уделили серьезное внимание обес- печению безопасности работы пользователей. Это подтверждается категориями, присвоенными различным версиям данной операционной системы по тем или иным международным и национальным критериям оценки безопасности. Так, по классификации «Оранжевой книги» ОС Windows NT 4 еще в 1999 году получила класс безопасности C2, по стандарту ISO/IEC 15408 Common Criteria for Information Technology Security Evaluation (Общие критерии оценки безопасности информаци- онных технологий) клиентские и серверные версии от Windows 2000 до Windows 10, от Windows Server 2008 до Windows Server 2013 получили уровень безопасности EAL4+, а операционные системы Windows 8, Windows Server 2012 Standard соот- ветствуют требованиям руководящего документа «Средства вычислительной тех- ники. Защита от несанкционированного доступа к информации. Показатели защи- щенности от несанкционированного доступа к информации» (Гостехкомиссия Рос- сии.1992) по 5 классу защищенности.

Какие средства безопасности предоставляют операционные семейства можно представить, если, например, познакомиться с требованиями оценки безопасности C2 «Оранжевой книги». Согласно им, система должна обеспечивать:

 средство безопасного входа в систему, которое обеспечивают точную идентификацию пользователей и предоставляют им возможности доступа к ресур- сам компьютера только после прохождения процедуры аутентификации. В Windows за идентификацию и аутентификацию пользователей отвечают процес- сы Winlogon.exe и Lsass.exe.

 управление доступом, позволяющее владельцу ресурса (файла, раздела реестра, объекта ядра и др.) определить, кто имеет права на доступ к ресурсу, а также уточнить суть этих прав (чтение, изменение, запуск и т.п.). При использова- нии дискреционной модели доступа для уплотнения матрицы доступа владелец может наделять правами, разрешающими различные виды доступа к объекту, как отдельного пользователя, так группу пользователей. Безопас- ный доступ реализуется в Windows компонентом Security Reference Monitor (SRM, монитор контроля безопасности) исполнительной системы Ntoskrnl.exe.

 аудит безопасности, позволяющий регистрировать события, относящи- еся к вопросам безопасности. Идентификация пользователей при входе в систему позволяет привязывать все события безопасности в системе к конкретному пользо- вателю. В Windows аудит поддерживается SRM и Lsass.exe.

 защита от повторного использования объекта, которая не позволяет пользователям просматривать данные, удаленные другим пользователем, или не

позволяет обращаться к памяти, которая ранее была использована, а затем осво- бождена другим пользователем. В Windows освобожденная память очищается системным потоком обнуления страниц, работающим во время простоя системы (с нулевым приоритетом).

1.2. Идентификация пользователей

Для защиты данных Windows использует следующие основные механизмы: аутентификация и авторизация пользователей, аудит событий в системе, шифрова- ние данных, поддержка инфраструктуры открытых ключей, встроенные средства сетевой защиты. Эти механизмы поддерживаются такими подсистемами Windows как LSASS (Local Security Authority Subsystem Service, подсистема локальной аутентификации), SAM (Security Account Manager, диспетчер локальных записей безопасности), SRM (Security reference Monitor, монитор состояния защиты), Active Directory (служба каталогов), EFS (Encrypting File System, шифрующая файловая система) и др.

Защита объектов и аудит действий с ними в ОС Windows организованы на основе избирательного (дискреционного) доступа, когда права доступа (чтение, за- пись, удаление, изменение атрибутов) субъекта к объекту задается явно в специ- альной матрице доступа. Для укрупнения матрицы пользователи могут объеди- няться в группы. При попытке субъекта (одного из потоков процесса, запущенного от его имени) получить доступ к объекту указываются, какие операции пользова- тель собирается выполнять с объектом. Если подобный тип доступа разрешен, по- ток получает описатель (дескриптор) объекта и все потоки процесса могут выпол- нять операции с ним. Подобная схема доступа, очевидно, требует аутентификации каждого пользователя, получающего доступ к ресурсам и его надежную идентифи- кацию в системе, а также механизмов описания прав пользователей и групп поль- зователей в системе, описания и проверки дискреционных прав доступа пользова- телей к объектам. Рассмотрим, как в ОС Windows организована аутентификация и авторизация пользователей.

Все действующие в системе объекты (пользователи, группы, локальные ком- пьютеры, домены) идентифицируются в Windows не по именам, уникальность ко- торых не всегда удается достичь, а по идентификаторам защиты (Security Identi- fiers, SID). SID представляет собой числовое значение переменной длины:

S – R – I – S0 - S1 - … - Sn – RID

S - неизменный идентификатор строкового представления SID;

R – уровень ревизии (версия). На сегодня 1.

I - (identifier-authority) идентификатор полномочий . Представляет собой 48- битную строку, идентифицирующую компьютер или сеть, который(ая) выдал SID объекту. Возможные значения:

- 0 (SECURITY\_NULL\_SID\_AUTHORITY) — используются для сравнений, когда неизвестны полномочия идентификатора;

- 1 (SECURITY\_WORLD\_SID\_AUTHORITY) — применяются для констру- ирования идентификаторов SID, которые представляют всех пользовате- лей. Например, идентификатор SID для группы Everyone (Все пользовате- ли) — это S-1-1-0;

- 2 (SECURITY\_LOCAL\_SID\_AUTHORITY) — используются для построе- ния идентификаторов SID, представляющих пользователей, которые вхо- дят на локальный терминал;

- 5 (SECURITY\_NT\_AUTHORITY) — сама операционная система. То есть, данный идентификатор выпущен компьютером или доменом.

Sn – 32-битные коды (колчеством 0 и более) субагентов, которым было пере- дано право выдать SID. Значение первых подчиненных полномочий общеизвестно. Они могут иметь значение:

- 5 — идентификаторы SID присваиваются сеансам регистрации для выдачи прав любому приложению, запускаемому во время определенного сеанса регистрации. У таких идентификаторов SID первые подчиненные полно- мочия установлены как 5 и принимают форму S-1-5-5-x-y;

- 6 —когда процесс регистрируется как служба, он получает специальный идентификатор SID в свой маркер для обозначения данного действия. Этот идентификатор SID имеет подчиненные полномочия 6 и всегда будет S-1- 5-6;

- 21 (SECURITY\_NT\_NON\_UNIQUE) — обозначают идентификатор SID пользователя и идентификатор SID компьютера, которые не являются уни- кальными в глобальном масштабе;

- 32 (SECURITY\_BUILTIN\_DOMAIN\_RID) — обозначают встроенные идентификаторы SID. Например, известный идентификатор SID для встро- енной группы администраторов S-1-5-32-544;

- 80 (SECURITY\_SERVICE\_ID\_BASE\_RID) — обозначают идентификатор SID, который принадлежит службе.

Остальные подчиненные полномочия идентификатора совместно обозначают домен или компьютер, который издал идентификатор SID.

RID – 32-битный относительный идентификатор. Он является является иден- тификатором уникального объекта безопасности в области, для которой был опре- делен SID. Например, 500 — обозначает встроенную учетную запись Administrator, 501 — обозначает встроенную учетную запись Guest, а 502 — RID для билета на получение билетов протокола Kerberos .

При генерации SID Windows использует генератор случайных чисел, чтобы обеспечить уникальность SID для каждого пользователя. Для некоторого произ- вольного пользователя SID может выглядеть так:

S-1-5-21-789336058-484763869-725345543-1003

Предопределенным пользователям и группам Windows выдает характерные SID, состоящие из SID компьютера или домена и предопределенного RID. В таб- лице 1 приведен перечень некоторых общеизвестных SID.

Таблица 1. Общеизвестные SID Windows

SID Название Описание

S-1-1-0 Все Группа, в которую входят все пользователи

S-1-5-2 Сеть Группа, в которую входят все пользовате- ли, зарегистрировавшиеся в системе из се- ти

S-1-5-7 Анонимный вход Группа, в которую входят все пользовате-

 ли, вошедшие в систему анонимно

S-1-5-домен-500 Администратор Учетная запись администратора системы. По умолчанию только эта запись обеспечи- вает полный контроль системы

S-1-5-домен-501 Гость Учетная запись пользователя-гостя

Полный список общеизвестных SID можно посмотреть в документации Platform SDK. Узнать SID конкретного пользователя в системе, а также SID групп, в которые он включен, можно, используя консольную команду whoami:

whoami /user /sid

Соответствие имени пользователя и его SID можно отследить также в ключе реестра HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ ProfileList.

После аутентификации пользователя процессом Winlogon, все процессы, за- пущенные от имени этого пользователя будут идентифицироваться специальным объектом, называемым маркером доступа (access token). Если процесс пользова- теля запускает дочерний процесс, то его маркер наследуются, поэтому маркер до- ступа олицетворяет пользователя для системы в каждом запущенном от его имени процессе. Основные элементы маркера представлены на рис. 1.

SID пользо- вателя SID1 … SIDn

Идентификаторы групп пользователя DACL по умолчанию Привилегии Прочие па- раметры

Рисунок 1. Обобщенная структура маркера доступа.

Маркер доступа содержит идентификатор доступа самого пользователя и всех групп, в которые он включен. В маркер включен также DACL по умолчанию - первоначальный список прав доступа, который присоединяется к создаваемым пользователем объектам. Еще одна важная для определения прав пользователя в системе часть маркера – список его привилегий. Привилегии - это права доверен- ного объекта на совершение каких-либо действий по отношению ко всей системе. В таблице 2 перечислены некоторые привилегии, которые могут быть предостав- лены пользователю.

Таблица 2. Привилегии, которыми могут быть наделены пользователи

Имя и идентифика- тор привилегии Описание привилегии

Увеличение приоритета диспетчирования

SeIncreaseBasePriorityPrivilege Пользователь, обладающий данной привилегией может изменять приоритет диспетчирования процесса с помощью интерфейса Дис- петчера задач

Закрепление страниц в памяти SeLockMemoryPrivilege Процесс получает возможность хранить данные в физической па- мяти, не прибегая к кэшированию данных в виртуальной памяти на

диске.

Управление аудитом и журналом безопасности SeAuditPrivilege Пользователь получает возможность указывать параметры аудита

доступа к объекту для отдельных ресурсов, таких как файлы, объ- екты Active Directory и разделы реестра.

Овладение файлами или иными объектами SeTakeOwnershipPrivilege Пользователь получает возможность становиться владельцем лю- бых объектов безопасности системы, включая объекты Active Directory, файлы и папки NTFS, принтеры, разделы реестра, служ-

бы, процессы и потоки

Завершение работы си- стемы

SeShutdownPrivilege Пользователь получает возможность завершать работу операцион- ной системы на локальном компьютере

Обход перекрестной проверки

SeChangeNotifyPrivilege Используется для обхода проверки разрешений для промежуточ- ных каталогов при проходе многоуровневых каталогов

Управление привилегиями пользователей осуществляется в оснастке

«Групповая политика», раздел Конфигурация Windows/Локальные полити- ки/Назначение прав пользователя.

Чтобы посмотреть привилегии пользователя, можно также использовать команду

whoami /all

Остальные параметры маркера носят информационный характер и опреде- ляют, например, какая подсистема создала маркер, уникальный идентификатор маркера, время его действия. Необходимо также отметить возможность создания ограниченных маркеров (restricted token), которые отличаются от обычных тем, что из них удаляются некоторые привилегии и его SID-идетификаторы проверяются только на запрещающие правила. Создать ограниченный маркер можно программ- но, используя API-функцию CreateRestrictedToken, а можно запустить процесс с ограниченным маркером, используя пункт контекстного меню Windows “Запуск от имени другого пользователя” (рис.2).

Рисунок 2. Запуск процесса с ограниченным маркером

Ограниченные маркеры используются для процессов, подменяющих клиен- та и выполняющих небезопасный код.

Маркер доступа может быть создан не только при первоначальном входе пользователя в систему. Windows предоставляет возможность запуска процессов от имени других пользователей, создавая для этих процессов соответствующий мар- кер. Для этих целей можно использовать:

- API-функции CreateProcessAsUser, CreateProcessWithLogon;

- оконный интерфейс (рис.2), инициализирующийся при выборе пункта кон- текстного меню “Запуск от имени другого пользователя”;

- консольную команду runas:

runas /user:имя\_пользователя program ,

где имя\_пользователя - имя учетной записи пользователя, которая будет использо- вана для запуска программы в формате пользователь@домен или домен\пользова- тель;

program – команда или программа, которая будет запущена с помощью учетной записи, указанной в параметре /user.

В любом варианте запуска процесса от имени другой учетной записи необхо- димо задать ее пароль.

1.3. Защита объектов системы.

Маркер доступа идентифицирует субъектов-пользователей системы. С другой стороны, каждый объект системы, требующий защиты, содержит описание прав доступа к нему пользователей. Для этих целей используется дескриптор безопас- ности (Security Descriptor, SD). Каждому объекту системы, включая файлы, прин- теры, сетевые службы, контейнеры Active Directory и другие, присваивается де- скриптор безопасности, который определяет права доступа к объекту и содержит следующие основные атрибуты (рис.3):

- SID владельца, идентифицирующий учетную запись пользователя-владельца объекта;

- пользовательский список управления доступом (Discretionary Access Control List, DACL), который позволяет отслеживать права и ограничения, установленные владельцем данного объекта. DACL может быть изменен пользователем, который указан как текущий владелец объекта.

- системный список управления доступом (System Access Control List, SACL), определяющий перечень действий над объектом, подлежащих аудиту;

- флаги, задающие атрибуты объекта.

Авторизация Windows основана на сопоставлении маркера доступа субъекта с дескриптором безопасности объекта. Управляя свойствами объекта, администрато- ры могут устанавливать разрешения, назначать право владения и отслеживать до- ступ пользователей.

SD

Объект

Список управления доступом содержит набор элементов (Access Control En- tries, ACE). В DACL каждый ACE состоит из четырех частей: в первой указывают- ся пользователи или группы, к которым относится данная запись, во второй – права доступа, а третья информирует о том, предоставляются эти права или отбираются. Четвертая часть представляет собой набор флагов, определяющих, как данная за- пись будет наследоваться вложенными объектами (актуально, например, для папок файловой системы, разделов реестра).

Если список ACE в DACL пуст, к нему нет доступа ни у одного пользователя (только у владельца на изменение DACL). Если отсутствует сам DACL в SD объек- та – полный доступ к нему имеют все пользователи.

Если какой-либо поток запросил доступ к объекту, подсистема SRM осу- ществляет проверку прав пользователя, запустившего поток, на данный объект, просматривая его список DACL. Проверка осуществляется до появления разреша- ющих прав на все запрошенные операции. Если встретится запрещающее правило хотя бы на одну запрошенную операцию, доступ не будет предоставлен.

Рассмотрим пример на рис.4. Процесс пытается получить доступ к объекту с заданным DACL. В маркере процесса указаны SID запустившего его пользователя, а также SID групп, в которые он входит. В списке DACL объекта присутствуют разрешающие правила на чтение для пользователя с SID=100, и на запись для группы с SID=205. Однако, в доступе пользователю будет отказано, поскольку раньше встречается запрещающее запись правило для группы с SID=201.

SID 78 \_W Deny

SID 201 \_WX Deny

Необходимо отметить, что запрещающее правило помещено в списке DACL на рисунке не случайно. Запрещающие правила всегда размещаются перед разре- шающими, то есть являются доминирующими при проверке прав доступа.

Для определения и просмотра прав доступа пользователей к ресурсам можно использовать как графические средства контроля, так и консольные команды. Стандартное окно свойств объекта файловой системы (диска, папки, файла) на вкладке Безопасность (рис. 5) позволяет просмотреть текущие разрешения для пользователей и групп пользователей, редактировать их, создавать новые или уда- лять существующие.

Запрещающие ACE Разрещающие ACE

Рисунок 5 – GUI-интерфейс Windows для изменения прав доступа к объектам

При определении прав доступа к объектам можно задать правила их наследо- вания в дочерних контейнерах. В окне дополнительных параметров безопасности

на вкладке Разрешения при выборе опции «Добавлять разрешения, наследуе- мые от родительских объектов» (рис. 6) можно унаследовать разрешения и огра- ничения, заданные для родительского контейнера, текущему объекту.

Унаследовать права от родителя

Передать права дочер- ним объектам

Рисунок 6 – Определение параметров наследования прав доступа к объектам

При выборе опции «Применять эти разрешения к объектам и контейнерам только внутри этого контейнера» разрешается передача определенных для объ- екта-контейнера правил доступа его дочерним объектам.

В этом же окне на вкладке Владелец допустимо узнать владельца объекта и заменить его. Владелец объекта имеет право на изменение списка его DACL, даже если к нему запрещен любой тип доступа. Администратор имеет право становиться владельцем любого объекта.

С учетом возможности вхождения пользователя в различные группы и неза- висимости определения прав доступа к объектам для групп и пользователей, зача- стую бывает сложно определить конечные права пользователя на доступ к объекту: требуется просмотреть запрещающие правила, определенные для самого объекта, для всех групп, в которые он включен, затем то же проделать для разрешающих правил. Автоматизировать процесс определения разрешенных пользователю видов доступа к объекту можно с использованием вкладки «Действующие разрешения» окна дополнительных параметров безопасности объекта (рис. 7).

Имя пользователя или группы

Разрешения на доступ к объекту для выбранного пользователя или группы

Для просмотра и изменения прав доступа к объектам в режиме командной строки предназначена команда icacls (cacls в Windows XP).

icacls имя\_файла [/t] [/e] [/c] [/g пользователь:разрешение] [/r пользователь

[...]] [/p пользователь:разрешение [...]] [/d пользователь [...]]

Назначения параметров команды приведены в таблице 3.

Таблица 3. Параметры команды cacls

<имя файла> Задаёт файл или папку, права доступа к которой необхо- димо просмотреть/изменить (допустимо использовать шаблоны с символами \* и ?).

/t Изменение избирательных таблиц контроля доступа (DACL) указанных файлов в текущем каталоге и всех под- каталогах

/e Редактирование избирательной таблицы управления до-

ступом (DACL) вместо ее замены

/c Заставляет команду продолжить изменение прав доступа при возникновении ошибки, связанной с нарушениями прав доступа

/g <пользователь | группа: разрешение> Предоставление прав доступа указанному пользователю

/r <пользователь | группа> Отнимает права доступа указанного пользователя.

/p <пользователь |

группа: разрешение> Заменяет права доступа указанного пользователя

/d <пользователь | группа> Отказывает в праве доступа указанному пользователю или группе

Для указания добавляемых или отнимаемых прав используются следующие значения:

F - полный доступ;

C - изменение (запись);

W - запись;

R - чтение;

N - нет доступа.

Рассмотрим несколько примеров.

icacls d:\test

Выдаст список DACL для папки test.

icacls d:\test /d ИмяКомпьютера\ИмяПользователя /e Запретит доступ к объекту для указанного пользователя. icacls d:\test /p ИмяКомпьютера\ИмяГруппы:f /e /t

Предоставит полный доступ к папке d:\test и ее подпапках всем для членов указанной группы.

Для программного просмотра и изменения списков DACL можно использо- вать API-функции AddAccessAllowedAce, AddAccessDeniedAce, SetSecurityInfo. Подробнее с этими функциями и примерами их использования можно ознакомить- ся в [3].

Рассмотренные способы работы с дискреционным списком доступа иллю- стрируют реализацию в Windows модели произвольного доступа. Но начиная с Windows Vista фирма Microsoft реализовала элементы мандатного доступа для кон- троля доступа к объектам. За этот уровень обеспечения безопасности отвечает Windows Integrity Control (WIC). Концепция WIC вторит уже рассмотренным выше принципами принудительного (мандатного) управления доступом и основана на построении доверительных отношений между объектами и управлении действиями с ними пользователей на основе их уровня доверия. Базовым понятием WIC явля- ется уровень целостности (integrity level) объекта. WIC присваивает контролируе- мым объектам один из шести доступных уровней целостности:

 Untrusted — анонимные процессы автоматически попадают в эту категорию.

 Low — стандартный уровень при работе с Интернетом. Если браузер Internet Explorer запущен в защищенном режиме, все файлы и процессы, ассоциированные с ним, назначаются в эту категорию. Некоторые папки, такие как, например, Temporary Internet Folder, также по умолчанию наделяются Низким уровнем дове- рия.

 Medium — в данном контексте работает большинство объектов. Ординарные пользователи получают Средний уровень, всем объектам присваивается данный уровень доступа, если не указан какой-либо иной.

 High — уровень, ассоциированный в системе с Администраторами. Объек- ты Высокоо уровня недоступны обычным пользователям.

 System — уровень для работы ядра операционной системы и его служб.

 Installer — вершина в иерархии уровней WIC. Его объекты могут изменять и удалять файлы всех предыдущих уровней.

Контроль по уровням целостности при доступе к объекту также осуществля-

ется на основе правил ACE. Но это специализированные ACE, которые начиная с

Windows Vista хранятся в списке SACL дескриптора безопасности объекта наряду с правилами аудита. Уровень целостности пользователя (процесса, выполняющегося от его имени) хранится в его токене безопасности. При доступе процесса к объекту монитор безопасности сравнивает уровень целостности в токене с уровнем целост- ности в дескрипторе объекта (в SACL). Система выдает права доступа в зависимо- сти от того выше или ниже уровень целостности субъекта по отношению к объек- ту, а также в зависимости от флагов политики целостности в соответствующей ACE объекта. Уровни целостности (IL) пользователя описываются в его идентифи- каторе безопасности, точнее – в его RID-части:

SID = S-1-16-0x0 - уровень Untrusted SID = S-1-16-0x1000 - уровень Low SID= S-1-16-0x2000 – уровень Medium SID= S-1-16-0x3000 – уровень High SID= S-1-16-0x4000 – уровень системы

Для изменения уровня целостности объектов можно использовать следую- щие инструменты:

- уже рассмотренную команду icacls с ключом /setintegitylevel. Например, вот так можно присвоить файлу низкий (l) уровень целостности:

icacls f:\temp /setintegritylevel l

- используя специальные утилиты Chml ("change mandatory label" ) для изме- нения уровня целостности файлов и папок, и Regil ("Registry integrity levels") для работы с уровнями целостности ключей реестра.

Изменить уровень целостности процесса можно, например, запустив его ути- литой psexec.exe с соответствующим ключом. Вот как можно запустить блокнот с высоким уровнем целостности:

psexec –h notepad.exe

Очевидно, что изменять уровень целостности запускаемых процессов потен- циально небезопасная операция, поэтом ее могут запускать только процессы, у ко- торых в маркере доступа установлена привилегия SeRelabelPrivilege.

Узнать, каким уровнем целостности обладает процесс можно, например, за- пустив утилиту ProcessExplorer из набора Sysinternals [6] (рис. 8).

Рисунок 8 – Уровень целостности запущенных процессов в интерфейсе ProcessExplorer

Необходимо отметить, что контроль уровней целостности имеет более высо- кий приоритет при проверке прав доступа к объекту перед дискреционной табли- цей.

1.4. Подсистема аудита.

Важный элемент политики безопасности – аудит событий в системе. ОС Windows ведет аудит событий по 9 категориям:

1. Аудит событий входа в систему.

2. Аудит управления учетными записями.

3. Аудит доступа к службе каталогов.

4. Аудит входа в систему.

5. Аудит доступа к объектам.

6. Аудит изменения политики.

7. Аудит использования привилегий.

8. Аудит отслеживания процессов.

9. Аудит системных событий.

Рассмотрим более подробно, какие события отслеживает каждая из категорий.

Аудит событий входа в систему

Аудит попыток пользователя войти в систему с другого компьютера или выйти из нее, при условии, что этот компьютер используется для проверки под- линности учетной записи.

Аудит управления учетными записями

Аудит событий, связанных с управлением учетными записями на компьюте- ре: создание, изменение или удаление учетной записи пользователя или группы; переименование, отключение или включение учетной записи пользователя; задание или изменение пароля.

Аудит доступа к службе каталогов

Аудит событий доступа пользователя к объекту каталога Active Directory, для которого задана собственная системная таблица управления доступом (SACL).

Аудит входа в систему

Аудит попыток пользователя войти в систему с компьютера или выйти из

нее.

Аудит доступа к объектам

Аудит событий доступа пользователя к объекту – например, к файлу, папке,

разделу реестра, принтеру и т. п., - для которого задана собственная системная таб- лица управления доступом (SACL).

Аудит изменения политики

Аудит фактов изменения политик назначения прав пользователей, политик аудита или политик доверительных отношений.

Аудит использования привилегий

Аудит попыток пользователя воспользоваться предоставленным ему правом.

Аудит отслеживания процессов

Аудиту таких событий, как активизация программы, завершение процесса, повторение дескрипторов и косвенный доступ к объекту.

Аудит системных событий

Аудит событий перезагрузки или отключения компьютера, а также событий, влияющих на системную безопасность или на журнал безопасности.

Решения об аудите конкретного типа событий безопасности принимаются в соответствии с политикой аудита локальной системы. Политика аудита, также называемая локальной политикой безопасности (local security policy), является ча- стью политики безопасности, поддерживаемой LSASS в локальной системе, и настраивается с помощью редактора локальной политики безопасности (Оснастка gpedit.msc, Конфигурация компьютера - Конфигурация Windows – Параметры безопасности – Локальные политики – Политика аудита, рис. 9).

Рисунок 9 – Конфигурация политики аудита редактора локальной политики безопасности

Для каждого объекта в SD содержится список SACL, состоящий из записей ACE, регламентирующих запись в журнал аудита удачных или неудачных попыток доступа к объекту. Эти АСЕ определяют, какие операции, выполняемые над объек- тами конкретными пользователями или группами, подлежат аудиту. Информация аудита хранится в системном журнале аудита. Аудиту могут подлежать как успеш- ные, так и неудачные операции. Подобно записям ACE DACL, правила аудита объ- ектов могут наследоваться дочерними объектами. Процедура наследования опре- деляются набором флагов, являющихся частью структуры ACE.

Настройка списка SACL может быть осуществлена в окне дополнительных свойств объекта (пункт “Дополнительно”, закладка “Аудит”, рис. 10).

записи ACE списка SACL объекта

параметры наследования ACE (аналогично DACL)

Рисунок 10 – Интерфейс редактирования правил аудита для объекта

Для программного просмотра и изменения списков SACL можно использовать API-функции GetSecutityInfo и SetSecutityInfo.

При инициализации системы и изменении политики LSASS посылает SRM сообщения, информирующие его о текущей политике аудита. LSASS отвечает за прием записей аудита, генерируемых на основе событий аудита от SRM, их редак- тирование и передачу Event Logger (регистратору событий). SRM посылает записи аудита LSASS через свое LPC-соединение. После этого Event Logger заносит запи- си в журнал безопасности.

Начиная с Windows Vista поддерживаются две категории журналов событий: журналы Windows и журналы приложений и служб. Журналы Windows – регистрируют общесистемных событий, и ведутся самой ОС. Журналы приложе- ний и служб – индивидуальны для конкретных типов приложений и компонент (Internet Explorer, MediaCenter, PoerShell и др.). События аудита записываются в журналы Windows следующих типов (на примере Windows 7):

1. Журнал приложений. В журнале приложений содержатся данные, относя- щиеся к работе приложений и программ.

2. Журнал безопасности. Журнал безопасности содержит записи о таких со- бытиях, как успешные и безуспешные попытки доступа в систему, а также о собы- тиях, относящихся к использованию ресурсов.

3. Журнал системы. В журнале системы содержатся события системных ком- понентов Windows. Например, в журнале системы регистрируются сбои при за- грузке драйвера или других системных компонентов при запуске системы.

4. Журнал установки. Фиксирует события, связанные с установкой или уда- лением компонент системы.

5. Журнал перенаправления. Фиксирует события, перенаправленные с со- седних компьютеров.

Просмотр журнала безопасности осуществляется в оснастке «Просмотр со- бытий» (eventvwr.msc, рис. 11). Сами журналы хранятся в файлах с расширением evtx в папке %SystemRoot%\System32\Winevt\Logs\.

Информационное сообщение Сообщение об ошиб- ке

Предупреждающее сообщение

В журнал записываются события различных типов:

- Сведение – сигнализирует об изменении в приложении или компоненте, например, успешном доступе к ресурсу, запуске приложения или службы;

- Предупреждение – сигнализирует о потенциально опасном событии, воз- никшем в приложении или компоненте, которые не мешают его работе, но могут стать причиной проблем в будущем;

- Ошибка – сигнализирует о проблему, сказывающемся на окружени прило- жения или компонента, вызвавших событие;

- Критическая ошибка - соответствует сбою, критичному для приложения или компонента, после которого они не могут продолжать работу;

1.5. Шифрующая файловая система.

Начиная с версии Windows 2000, в операционных системах семейства Windows NT поддерживается шифрование данных на разделах файловой системы NTFS с использованием шифрующей файловой системы (Encrypted File System, EFS). Основное ее достоинст во заключается в обеспечении конфиденциальности данных на дисках компьютера за счет использования надежных симметричных ал- горитмов для шифрования данных в реальном режиме времени.

Для шифрации данных EFS использует симметричный алгоритм шифрования (AES или DESX) со случайным ключом для каждого файла (File Encryption Key, FEK). По умолчанию данные шифруются в Windows 2000 и Windows XP по алго- ритму DESX, а в Windows XP с Service Pack 1 (или выше) и Windows Server 2003

— по алгоритму AES. В версиях Windows, разрешенных к экспорту за пределы США, драйвер EFS реализует 56-битный ключ шифрования DESX, тогда как в вер- сии, подлежащей использованию только в США, и в версиях с пакетом для 128- битного шифрования длина ключа DESX равна 128 битам. Алгоритм AES в Windows использует 256-битные ключи.

При этом для обеспечения секретности самого ключа FEK шифруется асим- метричным алгоритмом RSA открытым ключом пользователя, результат шифрации FEK – Data Decryption Field, DDF – добавляется в заголовок зашифрованного файла (рис. 12). Такой подход обеспечивает надежное шифрование без потери эф- фективности процесса шифрования: данные шифруются быстрым симметричным

алгоритмом, а для гарантии секретности симметричного ключа используется асим- метричный алгоритм шифрования.

Открытый ключ пользователя (RSA)

Файл на диске

FEK

FEK

Файл, зашифро- ванный EFS

Открытый ключ аген- та восстановления RSA)

Рисунок 12. Схема шифрации файла в EFS

Для шифрации файлов с использованием EFS можно использовать графиче- ский интерфейс или команду cipher.

Графический интерфейс доступен в стандартном окне свойств объекта по нажатию кнопки «Дополнительно» (рис. 13). Зашифрованные объекты в стан- дартном интерфейсе Windows Explorer отображаются зеленым цветом.

Необходимо отметить, что EFS позволяет разделять зашифрованный файл между несколькими пользователями. В этом случае FEK шифруется открытыми ключами всех пользователей, которым разрешен доступ к файлу, и каждый резуль- тат шифрации добавляется в DDF.

Шифрование файла с использованием EFS защищает файл комплексно: поль- зователю, не имеющему права на дешифрацию файла, недопустимы, в том числе, такие операции, как удаление, переименование и копирование файла. Необходимо помнить, что EFS является частью файловой системы NTFS, и в случае копирова-

ния защищенного файла авторизованным пользователем на другой том с файловой системой, на поддерживающей EFS (например, FAT32), он будет дешифрован и сохранен на целевом томе в открытом виде.

Консольная команда cipher может быть использована для шифра- ции/дешифрации файлов из командной строки или в bat-сценарии.

cipher [{/e|/d}] [/s:каталог] [/a] [/i] [/f] [/q] [/h] [/k] [/u[/n]] [путь [...]] | [/r:имя\_файла\_без\_расширения]

Назначения параметров команды приведены в таблице 4.

Таблица 4. Параметры команды cipher

/e Шифрует указанные папки. Папки помечаются таким образом, что-

бы файлы, которые будут добавляться в папку позже, также шифро- вались.

/d Расшифровывает указанные папки. Папки помечаются таким обра- зом, чтобы файлы, которые будут добавляться в папку позже, не бу-

дут шифроваться

/s: каталог Выполняет выбранную операцию над указанной папкой и всеми

подпапками в ней.

/a Выполняет операцию над файлами и каталогами

/i Продолжение выполнения указанной операции даже после возник-

новения ошибок. По умолчанию выполнение cipher прекращается после возникновения ошибки

/f Выполнение повторного шифрования или расшифровывания ука-

занных объектов. По умолчанию уже зашифрованные или расшиф- рованные файлы пропускаются командой cipher

/k Создание ключа шифрования файла для пользователя, выполнивше- го команду cipher. Если используется данный параметр, все осталь-

ные параметры команды cipher не учитываются.

/u Обновление ключа шифрования файла пользователя или ключа агента восстановления на текущие ключи во всех зашифрованных файлах на локальном диске (если эти ключи были изменены). Этот

параметр используется только вместе с параметром /n.

/n Запрещение обновления ключей. Данный параметр служит для по- иска всех зашифрованных файлов на локальных дисках. Этот пара-

метр используется только вместе с параметром /u.

путь Указывает шаблон, файл или папку.

/r: имя\_файла Создание нового сертификата агента восстановления и закрытого ключа с последующей их записью в файлах с именем, указанным в параметре имя\_файла (без расширения). Если используется данный параметр, все остальные параметры команды cipher не учитывают-

ся.

Например, чтобы определить, зашифрована ли какая-либо папка, необходимо использовать команду:

cipher путь\имя\_папки

Команда cipher без параметров выводит статус (зашифрован или нет) для всех объектов текущей папки.

Для шифрации файла необходимо использовать команду

cipher /e /a путь\имя\_файла

Для дешифрации файла, соответственно, используется команда

cipher /d /a путь\имя\_файла

Допустима шифрация/дешифрация группы файлов по шаблону:

cipher /e /a d:\work\\*.doc

Пара открытый и закрытый ключ для шифрации FEK создаются для пользо- вателя автоматически при первой шифрации файла с использованием EFS.

Если некоторый пользователь или группа пользователей зашифровали файл с использованием EFS, то его содержимое доступно только им. Это приводит к рис- кам утери доступа к данным в зашифрованных файлах в случае утраты пароля дан- ным пользователем (работник забыл пароль, уволился и т.п.). Для предотвращения подобных проблем администратор может определить некоторые учетные записи в качестве агентов восстановления.

Агенты восстановления (Recovery Agents) определяются в политике без- опасности Encrypted Data Recovery Agents (Агенты восстановления шифро- ванных данных) на локальном компьютере или в домене. Эта политика доступна через оснастку Групповая политика (gpedit.msc) раздел «Параметры безопасно- сти»-> «Политика открытого ключа»-> «Файловая система EFS». Пункт меню

«Действие»-> «Добавить агент восстановления данных» открывает мастер до- бавления нового агента.

Добавляя агентов восстановления можно указать, какие криптографические пары (обозначенные их сертификатами) могут использовать эти агенты для восста- новления шифрованных данных (рис. 14). Сертификаты для агентов восстановле- ния создаются командой cipher с ключом /r (см. табл. 4). Для пользователя, кото- рый будет агентом восстановления, необходимо импортировать закрытый ключ агента восстановления из сертификата, созданного командой cipher. Это можно сделать в маcтере импорта сертификатов, который автоматически загружается при двойном щелчке по файлу \*.pfx.

EFS создает DRF (Data Recovery Field) – элементы ключей для каждого агента восстановления, используя провайдер криптографических сервисов, зареги- стрированный для EFS-восстановления. DRF добавляется в зашифрованный файл и может быть использован как альтернативное средство извлечения FEK для дешиф- рации содержимого файла.

Windows

хранит закрытые ключи в подкаталоге Application Data\Micro-soft\Crypto\RSA каталога профиля пользователя. Для защиты закрытых ключей Windows шифрует все файлы в папке RSA на основе симметричного ключа, генерируемого случай- ным образом; такой ключ называется мастер-ключом пользователя. Мастер-ключ имеет длину в 64 байта и создается стойким генератором случайных чисел. Ма- стер-ключ также хранится в профиле пользователя в каталоге Application Data\Microsoft\Protect и зашифровывается по алгоритму 3DES с помощью ключа, который отчасти основан на пароле пользователя. Когда пользователь меняет свой пароль, мастер-ключи автоматически расшифровываются, а затем заново зашифро- вываются с учетом нового пароля.

Для расшифровки FEK EFS использует функции Microsoft CryptoAPl (CAPI). CryptoAPI состоит из DLL провайдеров криптографических сервисов (cryptographic service providers, CSP), которые обеспечивают приложениям доступ к различным криптографическим сервисам (шифрованию, дешифрованию и хэшированию). EFS опирается на алгоритмы шифрования RSA, предоставляемые провайдером Microsoft Enhanced Cryptographic Provider.

Шифрацию и дешифрацию файлов можно осуществлять программно, ис- пользуя API-функции EncryptFile и DecryptFile.

2. Порядок выполнения работы.

2.1. Ознакомьтесь с теоретическими основами защиты информации в ОС се- мейства Windows в настоящих указаниях и конспектах лекций.

2.2. Выполните задания 2.2.1-2.2.8

2.2.1. При выполнении лабораторной работы на компьютерах в учеб- ной лаборатории запустите в программе Oracle VM Virtualbox виртуальную ма- шину Win7Test. Войдите в систему под учетной записью администратора, пароль узнайте у преподавателя. Все действия в пп 2.2.1-2.2.8 выполняйте в системе, рабо- тающей на виртуальной машине.

2.2.2. Создайте учетную запись нового пользователя testUser в оснаст- ке «Управление компьютером» (compmgmt.msc). При создании новой учетной

записи запретите пользователю смену пароля и снимите ограничение на срок дей- ствия его пароля. Создайте новую группу ”testGroup” и включите в нее нового пользователя. Удалите пользователя из других групп. Создайте на диске С: папку forTesting. Создайте или скопируйте в эту папку несколько текстовых файлов (\*.txt).

2.2.3. С помощью команды runas запустите сеанс командной строки (cmd.exe) от имени вновь созданного пользователя. Командой whoami посмотрите SID пользователя и всех его групп, а также текущие привилегии пользователя. Строку запуска и результат работы этой и всех следующих консольных команд копируйте в файл протокола лабораторной работы.

2.2.4. Убедитесь в соответствии имени пользователя и полученного SID в реестре Windows. Найдите в реестре, какому пользователю в системе присвоен SID S-1-5-21-1957994488-492894223-170857768-1004 (Используйте ключ реестра HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList).

2.2.5. Командой whoami определите перечень текущих привилегий пользователя testUser. В сеансе командной строки пользователя попробуйте изме- нить системное время командой time. Чтобы предоставить пользователю подобную привилегию, запустите оснастку «Локальные параметры безопасности» (secpol.msc). Добавьте пользователя в список параметров политики «Изменение системного времени» раздела Локальные политики -> Назначение прав поль- зователя. После этого перезапустите сеанс командной строки от имени пользова- теля, убедитесь, что в списке привилегий добавилась SeSystemtimePriviege. По- пробуйте изменить системное время командой time.

Убедитесь, что привилегия «Завершение работы системы» (SeShutdown- Privilege) предоставлена пользователю testUser . После этого попробуйте завер- шить работу системы из сеанса командной строки пользователя командой shut- down –s. Добавть ему привелегию «Принудительное удаленное завершение» (SeRemoteShutdownPrivilege). Попробуйте завершить работу консольной коман- дой еще раз (отменить команду завершения до ее непосредственного выполнения можно командой shutdown –a).

2.2.6. Ознакомьтесь с справкой по консольной команде icacls. Используя эту команду, просмотрите разрешения на папку c:\forTesting. Объясните все обозначе- ния в описаниях прав пользователей и групп в выдаче команды.

а) Разрешите пользователю testUser запись в папку forTesting, но запретите запись для группы testGroup. Попробуйте записать файлы или папки в forTesting от имени пользователя testUser. Объясните результат. Посмотрите эффективные разрешения пользователя testUser к папке forTesting в окне свойств папки.

б) Используя стандартное окно свойств папки, задайте для пользователя tes- tUser такие права доступа к папке, чтобы он мог записывать информацию в папку forTesting, но не мог просматривать ее содержимое. Проверьте, что папка forTesting является теперь для пользователя testUser “слепой”, запустив, напри- мер, от его имени файловый менеджер и попробовав записать файлы в папку, про- смотреть ее содержимое, удалить файл из папки.

в) Для вложенной папки forTesting\Docs отмените наследование ACL от родителя и разрешите пользователю просмотр, чтение и запись в папку. Проверьте, что для пользователя папка forTesting\Docs перестала быть “слепой” (например,

сделайте ее текущей в сеансе работы файлового менеджера от имени пользователя и создайте в ней новый файл).

г) Снимите запрет на чтение папки forTesting для пользователя testUser.

Используя команду icacls запретите этому пользователю доступ к файлам с расши- рением txt в папке forTesting. Убедитесь в недоступности файлов для пользовате- ля.

д) Командой icacls запретите пользователю все права на доступ к папке forTesting и разреште полный доступ к вложенной папке forTesting\Docs. Убеди- тесь в доступности папки forTesting\Docs для пользователя. Удалите у пользовате- ля testUser привилегию SeChangeNotifyPrivilege. Попробуйте получить доступ к папке forTesting\Docs. Объясните результат.

е) Запустите файловый менеджер от имени пользователя testUser и создайте в нем папку newFolder на диске C. Для папки newFolder очистите весь список ACL командой cacls. Попробуйте теперь получить доступ к папке от имени адми- нистратора и от имени пользователя. Кто и как теперь может вернуть доступ к пап- ке? Верните полный доступ к папке для всех пользователей.

ж) Создайте в разделе HKLM\Software реестра раздел testKey. Запретите пользователю testUser создание новых разделов в этом разделе реестра. Создайте для раздела HKLM\Software\testKey SACL, позволяющий протоколировать отка- зы при создании новых подразделов, а также успехи при перечислении подразде- лов и запросе значений (предварительно проверьте, что в локальной политике без- опасности соответствующий тип аудита включен). Попробуйте от имени пользова- теля testUser запустить regedit.exe и создать раздел в HKLM\Software. Убедитесь, что записи аудита были размещены в журнале безопасности (eventvwr.msc).

з) С использованием команды whoami проверьте уровень целостности для пользователя testUser и администратора (учетная запись ВПИ). Запустите какое- нибудь приложение (калькулятор, блокнот) от имени testUser и администратора. С использованием утилиты ProcessExplorer (можно найти в папке c:\Utils на вирту- альной машине) проверьте уровень целостности запущенных приложений. Объяс- ните разницу. Верните пользователю testUser права на полный доступ к папке forTesting. От имени администратора создайте в папке forTesting текстовый файл someText.txt. Измените уровень целостности этого файла до высокого с использо- ванием команды icacls. Запустите блокнот от имени пользователя testUser, открой- те в нём файл someText.txt, измените содержимое файла и попробуйте сохранить изменения. Объясните причину отказа в доступе. Как можно предоставить пользо- вателю testUset доступ к файлу?

2.2.7. Шифрование файлов и папок средствами EFS.

а) От имени пользователя testUser зашифруйте какой-нибудь файл на диске. Убедитесь, что после этого был создан сертификат пользователя, запустив оснаст- ку certmgr.msc от имени пользователя (раздел Личные). Просмотрите основные параметры сертификата открытого ключа пользователя testUser (срок действия, используемые алгоритмы). Установите доверие к этому сертификату в вашей си- стеме.

б) Создайте в папке forTesting новую папку Encrypt. В папке Encrypt со- здайте или скопируйте в нее текстовый файл. Зашифруйте папку Encrypt и все ее содержимое из меню свойств папки от имени администратора. Попробуйте про-

смотреть или скопировать какой-нибудь файл этой папки от имени пользователя testUser. Объясните результат. Скопируйте зашифрованный файл в незашифро- ванную папку (например, forTesting). Убедитесь что он остался зашифрованным. Добавьте пользователя testUser в список имеющих доступа к файлу пользователей в окне свойств шифрования файла. Повторите попытку получить доступ к файлу от имени пользователя testUser.

в) Создайте учетную запись нового пользователя agentUser, сделайте его членом группы Администраторы. Определите для пользователя agentUser роль агента восстановления EFS. Создайте в папке forTesting новый текстовый файл с произвольным содержимым. Зашифруйте этот файл от имени пользователя testUser. Убедитесь в окне подробностей шифрования файла, что пользователь agentUser является агентом восстановления для данного файла. Попробуйте про- читать содержимое файла от имени администратора и от имени пользователя agentUser. Объясните результат.

г) Зашифруйте все текстовые файлы папки forTesting с использованием кон- сольной команды шифрования cipher от имени пользователя testUser (предвари- тельно снимите запрет на доступ к этим файлам, установленный в задании 2.2.6г).

д) Убедитесь, что при копировании зашифрованных файлов на том с файло- вой системой, не поддерживающей EFS (например, FAT32 на флеш-накопителе), содержимое файла дешифруется.

2.2.8. После демонстрации результатов работы преподавателю восстановите исходное состояние системы: удалите созданные папки и файлы, разделы реестра, удалите учетную запись созданного пользователя и его группы, снимите с пользо- вателя agentUser роль агента восстановления.

2.2.9. Представьте отчёт по лабораторной работе преподавателю и отчитайте работу.

2.3. Содержание отчета

Отчет по лабораторной работе должен содержать следующие сведения:

- название и цель работы;

- протокол выполнения лабораторной работы, содержащий список кон- сольных команд, составленных при выполнении работы, и результаты их выполнения.

3. Контрольные вопросы

1. К какому классу безопасности относится ОС Windows по различным критериям оценки?

2. Каким образом пользователи идентифицируются в ОС Windows?

3. Что такое списки DACL и SACL?

4. Перечислите, каким образом можно запустить процесс от имени друго- го пользователя.

5. Как происходит проверка прав доступа пользователя к ресурсам в ОС Windows?

6. Что такое маркер безопасности, и какова его роль в модели безопасно- сти Windows?

7. Как с использованием команды icacls добавить права на запись для всех файлов заданной папки?

8. Что такое уровень целостности? Как он влияет на права доступа субъектов к объектам ОС? Как можно узнать и задать уровень целостности для объектов и субъектов?

9. Какие события подлежат аудиту в ОС Windows?

10. Каким образом шифруются файлы в файловой системе EFS? Что такое FEK? DDF? DDR?

11. Какие алгоритмы шифрования используются в EFS?

**Практическое занятие №23 Асимметричная криптография и электронная цифровая подпись на примере си- стемы GnuPG**

Цель работы: знакомство с принципами криптографической защиты информации с использованием алгоритмов асимметричного шифрования и электронной циф- ровой подписи, приобретение навыков практического применения указанных ме- тодов защиты информации на основе системы GnuPG.

1. Основные принципы асимметричной криптографии

Долгое время шифрование как способ преобразования сообщения в форму, недоступную для восприятия неавторизованным пользователям, существовало только в форме симметричной криптографии, когда и отправитель, и получатель должны знать секретный ключ, используемый для шифрации/дешифрации сооб- щений.

Симметричное шифрование имеет недостатки, которые ограничивают воз- можности его применения в ряде конкретных случаев [2]. В частности, зачастую невозможно организовать секретный канал для обмена ключами шифрования между участниками взаимодействия. Еще одним недостатком симметричных шифров является необходимость хранения большого количества ключей. Для того чтобы в вычислительной сети могли конфиденциально попарно взаимодейство- вать N участников, необходимо наличие в системе N\*(N-1)/2 ключей. Эти недо- статки можно устранить, используя алгоритмы асимметричного шифрования. Например, для асимметричной системы достаточно иметь 2\*N пар откры- тый/закрытый ключ, чтобы можно было организовать секретный канал между каждой парой участников.

Асимметричная система шифрования работает по схеме, представленной на рис. 1. Отличительной особенностью асимметричных алгоритмов является нали- чие пары ключей шифрования: открытого (публичного) kот, который передается второй стороне по незащищенному каналу связи и поэтому может быть известен криптоаналитику, а также закрытого (частного) kзак, который известен лишь од- ному человеку (получателю сообщения) и держится в секрете [1]. Пара ключей обладает тем свойством, что сообщение, зашифрованное на одном из ключей, может быть расшифровано только на другом ключе. Фактически это означает, что секретным каналом передачи информации на схеме рис. 1 является направление “A-B”, поскольку сообщение, зашифрованное на открытом ключе отправителем, может дешифровать своим закрытым ключом только получатель. Если необходи- мо организовать двунаправленный безопасный обмен сообщениями, необходимо использовать две пары ключей и пользователь B должен шифровать информацию с использованием открытого ключа пользователя A.

Асимметричная система решает указанные выше проблемы симметричного шифрования – здесь нет необходимости передавать секретный ключ на противо- положную сторону, публичный ключ можно передавать по открытому каналу свя- зи. Единственное требование к каналу распространению открытого ключа – он

должен быть аутентичным, т.е. всякий, получивший этот ключ, должен иметь возможность убедиться в его принадлежности лицу, заявленному как владелец соответствующей ключевой пары. На практике это требование реализуется путем сертификации открытого ключа – третья сторон, которой доверяют обе стороны взаимодействия, заверяет открытый ключ и выдает на него сертификат, подпи- санный электронной цифровой подписью. Открытый ключ в этом случае распро- страняется вместе с сертификатом, что дает возможность всегда удостоверится в его принадлежности. К асимметричным относятся такие алгоритмы шифрования как RSA, El Gemal, Рабина, Месси-Омуры.

X-открытый текст

Y-шифротекст

E – процедура шифрации

D – процедура дешифрации

kот – открытый ключ

kзак – закрытый ключ

Сообщение

S

Асимметричная криптография не вытесняет полностью симметричные ал- горитмы с поля шифрования данных, поскольку сама обладает рядом недостатков, ключевой из которых – низкая скорость преобразования: асимметричный шифр в среднем на 3 порядка медленнее шифрует сообщение по сравнению с симметрич- ным при аналогичной криптостойкости. В связи с этим сегодня приоритетной яв- ляется гибридная схема шифрации, когда сообщение кодируется симметричным шифром с использованием сеансового ключа, а по асимметричной схеме шифру- ется и передается на противоположную сторону лишь сам сеансовый ключ.

Глядя на схему на рис.1 можно сделать вывод, что шифрация информации c использованием закрытого ключа не имеет смысла. Действительно, зачем кодиро- вать информацию, если всякий, кто знает открытый ключ (читай – любой желаю- щий) может ее декодировать. Однако, шифрование информации закрытым клю- чом имеет смысл. Тут необходимо вспомнить, что защита информации обеспечи- вает не только конфиденциальность информации, но и ее аутентичность, целост- ность, апеллируемость. Действительно, если мы зашифруем информацию закры-

тым ключом, а затем сможем дешифровать открытым, то сможем сделать вывод о том, что именно владелец пары ключей и никто другой шифровал информацию (закрытый ключ известен только ему), то есть она становится для нас аутентич- ной, а в случае использования сертифицированных ключей еще и апеллируемой. Это свойство ключевой пары лежит в основе формирования и верификации элек- тронной цифровой подписи (ЭЦП).

ЭЦП – это набор методов, которые позволяют перенести свойства рукопис- ной подписи под документом в область электронного документооборота [4]. Она обеспечивает аутентичность автора сообщения, уникальность подписи, контроль целостности передаваемого сообщения, невозможность переноса подписи под другой документ. Все эти свойства достигаются за счет синтеза асимметричной криптографии и хеширования документов.

Хеширование – процесс получения уникального дайджеста сообщения, ко- торый уникально идентифицирует сообщение: вычислительно трудно подобрать как сообщение с заранее известным хеш-дайджестом, так и два разных сообщения с одинаковым дайджестом. Формируют подобные дайджесты специальные хеш- функции. Примерами таких функция являются MD-5, SHA-1, SHA-2, SHA-3, RIPEMD-160, ГОСТ 34.11–12. На вход подобной функции подается сообщение произвольной длины, а на выходе формируется блок данных фиксированной дли- ны (от 128 до 512 бит у различных алгоритмов), который и будет уже упоминав- шимся дайджестом сообщения. Хеш-функция чувствительна ко входу и измене- ние даже одного бита во входном сообщении приводит к существенному (до 50% инверсии бит) изменению результирующего дайджеста на выходе. Поэтому очень трудно подобрать сообщение, которое будет иметь заданный дайджест. Рассмот- рим, как выглядит схема формирования и верификации ЭЦП (рис. 2).

В схеме на рис. 2 владелец пары ключей выступает отправителем письма с подписью. Чтобы сформировать подпись, отправитель получает дайджест текста сообщения и шифрует его своим закрытым ключом. Результат шифрования мож- но использовать как электронную подпись, ее вместе с самим сообщением от- правляют получателю. На стороне получателя подпись дешифруется открытым ключом отправителя и результат дешифрации сравнивается с результатом хеши- рования пришедшего сообщения. Если дайджест сообщения на стороне получате- ля совпадает с расшифрованным дайджестом стороны отправителя, то можно считать подпись верной. Подобную схему реализуют алгоритмы ЭЦП DSA, Шнорра, ГОСТ 34.10–12.

Для того, чтобы подменить письмо, злоумышленнику необходимо либо по- добрать другое, устраивающее его по содержимому, сообщение, которое имеет дайджест, аналогичный дайджеста исходного сообщения, либо по известному от- крытому ключу подобрать закрытый ключ и подменить в посылке как само пись- мо, так и подпись к нему. Обе эти задачи при условии использовании перечислен- ных выше алгоритмов ЭЦП является вычислительно неразрешимыми на совре- менном уровне технологий. Открытый и закрытый ключи генерируются одновре- менно, и между ними существует определенная математическая связь. Основная задача проектировщика асимметричного алгоритма заключается в том, чтобы по

известному открытому ключу было бы невозможно (очень трудоемко) получить секретный ключ шифрования. Для этого в основу асимметричных

алгоритмов закладываются вычислительно трудные задачи факторизации, дис- кретного логарифмирования, проецирования точек на эллиптической кривой и т.д. Практические аспекты использования асимметричной криптографии и элек- тронной цифровой подписи рассмотрим на примере использования программной системы GnuPG.

2. Общие сведения о GnuPG.

2.1. GnuPG – первое знакомство

GnuPG (англ. GNU Privacy Guard) — свободная распространяемая компью- терная система (распространяется под лицензией GNU General Public License), позволяющая выполнять операции шифрования (кодирования) и цифровой под- писи почтовых сообщений, файлов и другой информации, представленной в элек- тронном виде. Является идеологическим наследником разработанной Филиппом Циммерманном в 1991 году системы PGP, ставшей в середине 2000-х годов про- пиетарной. GPG предоставляет своим пользователям невскрывемые на современ- ном уровне развития криптологии криптоалгоритмы. Перечислим основные воз- можности системы:

 Полностью реализует стандарт OpenPGP.

 Поддерживает электронную подпись с помощью алгоритмов ElGamal, DSA, RSA и хеш-функций MD5, SHA-1, SHA-2, RIPE-MD-160 и TIGER.

 Поддерживает асимметричное шифрование с использованием алгоритмов ElGamal и RSA и длиной ключа от 1024 до 4096 бит.

 Позволяет осуществлять симметричное шифрование с использованием блочных алгоритмов AES, CAST5, 3DES, Twofish, Blowfish, Camellia.

 Поддерживает алгоритмы сжатия: ZIP, ZLIB, BZIP2.

 Имеет модульную архитектуру, позволяющую устанавливать плагины с до- полнительной функциональностью.

 Интегрированная поддержка серверов ключей.

 Может работать в консольном и графическом (для Window-платформ) ре- жиме.

GPG имеет множество реализаций, совместимых между собой и рядом дру- гих программ (например, PGP) благодаря стандарту OpenPGP (RFC 4880), но имеющих разный набор функциональных возможностей. Начиная с версии 2.0 GPG поддерживает стандарт S/MIME (IETF 3851, ITU-T X.509). Это позволяет поддерживать в рамках системы две системы сертификации – сетевую (OpenPGP) и иерархическую (S/MIME). Существуют реализации GPG для всех наиболее рас- пространённых операционных систем. Дистрибутивы системы можно скачать с сайта проекта https://www.gnupg.org.

2.2. Изготовление сертификатов ключей

Рассмотрим обобщенный порядок работы с системой GPG с использо- ванием стандарта OpenPGP. Выполнение криптографических действий над информацией невозможно без наличия ключей шифрования. Для генерации пар открытый/закрытый ключ система предлагает использовать сервер серти- фикации Cleopatra или консольную команду gpg с набором параметров запус- ка. Рассмотрим процесс генерации ключей с использованием Cleopatra. На рис. 3 представлен внешний вид программы.

Рисунок 3. Окно администратора ключей Cleopara

Непосредственно после установки в системе не зарегистрировано сер- тификатов ключей, чтобы они появились в окне необходимо либо создать, либо импортировать сертификаты. Процесс создания новой пары ключей инициируется выбором пункта меню File->New Certificate. В открывающемся окне необходимо выбрать тип используемых сертификатов ключей – Open- PGP (PGP/MIME) или X.509 (S/MIME).

Рисунок 4. Выбор типа генерируемых ключей

Рассмотрим процесс генерации ключей на примере стандарта OpenPGP. После выбора соответствующей опции и нажатия клавиши Next начинается работа мастера, который управлеят процессом создания ключей, предлагая пользователю сделать выбор в последовательности диалоговых окон. (рис.5). Первое окно предлагает задать персональные данные владельца пары ключей (имя, адрес электронной почты и комментарий, рис. 5а). Нажав кнопку Advanced Settings этого окна можно задать параметры генерируемой ключевой пары (рис. 5б) – размер ключей от 1536 до 4096 бит, алгоритм, который будет использоваться для шифрации и верификации ЭЦП, дату истечения срока действия ключевой пары, для каких целей (шифрование, ЭЦП, сертификация, аутентификация) она будет использоваться. Перед началом непосредственной генерации мастер предлагает еще раз посмотреть выбранные параметры генерируемых колючей (рис. 5в).

После нажатия кнопки CreateKey в этом окне начинается непосредственный процесс генерации, помочь в котором пользователь может, вводя случайные символы в окне, изображенное на рис.3г, для определения базы генерации случайных чисел для создаваемых ключей. Наконец, сстема предложит ввести парольную фразу (рис 5д), которая в дальнейшем будет защищать пользователя от несанкционированных манипуляцих стронних лиц с его закрытым ключем. Когда парольная фраза будет введена и корректно подтверждена, генератор создаст пару ключей и выведет окно об успешном завершении опреации (рис. 5е).

а) Ввод информации о владельце б) Определение параметров ключей

в)Поверка введенных данных г)Атоматизированная генерация случайных чисел для формирования ключа

д) Ввод парольной фразы для защиты е) Отчет об окончании генерации ключе зарыто- го ключа вой пары

Рисунок 5. Окна мастера генерации ключей

В этом окне отображается 40-цифровой fingerprint («отпечаток пальца») созданного сертификата ключевой пары, уникальность которого гаарантируется с очень высокой доли вероятности, что позволяет использловать этот отпечаток в качестве уникального в мировом масштабе идентификатора сгенерированной ключевой пары. Обычто в качестве

одентификатора сертификата использлуются последние 8 цифр отпечатка, что все равно гарантирует низкую вероятность коллизии именования разных ключевых пар.

По окончании процедуры генерации в окне Cleoptra вы можете увидеть строку, соответствующую вновь созданному сертификату (рис.6).

Рисунок 6. Сертификат ключевой пары отображается в окне менеджера ключей.

2.3. Распространение ключей.

Основной принцип асимметричной криптографии – хранить в секрете закрытый ключ для целей формирования ЭЦП и расшифровки пришедших сообщений и распространять публичный ключ своим клиентам, коллегам и собеседникам, чтобы они могли проверить вашу подпись под документом или зашифровать направляемое вам сообщение. Это делает актуальной задачу предоставления доступа к вашему публичному ключу всех желающих. Суще- ствуют два пути реализации этой задачи:

1) Через файл экспорта публичного ключа.

2) С использованием сервера сертификатов.

В первом случае необходимо воспользоваться командой File->Export Certificates. В ответ на эту команду необходимо выбрать папку и имя файла, в который будет экспортирован сертификат открытого ключа. Если выбрать в качестве расширения файла сертификата .asc, то сертификат будет экспор- тирован в текстовом виде:

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v2

mQENBFYOiIoBCACuwNxDPNiFu+DiVb6r4bmBCHmycOcce8JkQSHpwGlZcWdIZGZZ V34pEt4gnKBVBkDLchZDriyODRRYfbXyzaZ5kg5U/Lk5yWE6TVh4/mmhgvzX1A0w G4CgMD5Nfc+1KIztNgovOqmj7rTs8UpuZfzYNh06R8godvyX9WuVXLiAGxKeEhUS KZ8T8qIq137brgHpH8HtKtdZuCp8r0lo22O4XzQ9a/24jXt3fu/rt5Emf2BkI7H3 3QfHxtYjVRlsAreq8mJEEKI7v04xMiGvaJUQpcCOB3scyF4YRk6kQR3AIsXoedGV gPEcXg+ki/0JiQQi2Uq+OF2JrZmK6O6aH8tXABEBAAG0JnRlc3RrZXkgKHRlc3Qg a2V5IHBhaXIpIDx0ZXN0QG1haWwucnU+iQE/BBMBCAApBQJWDoiKAhsDBQkDw3eG BwsJCAcDAgEGFQgCCQoLBBYCAwECHgECF4AACgkQh2V6T6Cf8FNx1ggAlquzKSQU

OoXIfScM3vyjulj25Jlaga5LcxZ/wlh4eaSUnJIpAA+KTe/gUDYX0WMSDtoxYPXw z5wYslpGwA4VXhyv/Qhoyv+y/Ue3M9oXqyJf8nX5FQodbC+hHQFLbUibeg90UgWt RM7I7QuQxmJ3VeRlu5QxTNqhUrxKK8sJ4zuPdSoAlWSoI+Oey4OLj6e/sCEkCqEf bzKwjXhEScMje5RDCKbcx9AyHK90tsZf0qvNCj4K+7KZvqKAZa7Vyhdg7cIVPbqq KnfbBJb+1mrIAnKc4gOyDOCiMmDc5yQRVyhojrdaZeYepzFJgDFBf8jCQW1hphie b2Wa9XP+ipGReLkBDQRWDoiKAQgAs3pcXCNjJ6TWV2x4eTXnKW1FECgzAA1ZTfou KUVGu840yXKigTs5+2okjLVLkY0095s74Bmsxm2/h1DngjEV718PylVhFx4KISUS 3rQj9lZ2soF0hg3v7wtol8PPCm0unnxzb4HJtJ5PZg0QV5fBPld01eajQzbB8qaI CaIFcx+/wNE8O4EoWkhCix+EpESLxQfF8OKLEly12VEOrXMWGtJ393AtNCmuU0U8 kNlTj5mxp35vS7LSg4vzrFHNQkmCxUhijRfqwuMbU81t2/MTlz86Ul6nNFh/k/pF j9yHhPAhv8HQ/JlUE9Liwgj6t3K+wOpBsE6RNUANCz2V/p8VmwARAQABiQElBBgB CAAPBQJWDoiKAhsMBQkDw3eGAAoJEIdlek+gn/BTcw8H/0IIby3+gZTmZeFdQ+ys doimMkSfJ3QqWv0JPKcRnZiQ5rJpjv2MYctdBrkxUVv5Xvk2RBrwfU1tn4J5nyi8 ywEbpQMeRlj0ke61s4O03b2Gnf/oAemilqLZW4TwN17mppXqk8wHyuZHUVewpu3q iyKxMht9D9kfIUuV6l4sK6kz+oc+mfmp3AU19LZf71+WET8B4rg1q/Jc4zjQqvxZ jT/C0HE1sB8U1WbVX9IqEmHGy7JkVupATN5TrMl/og7wabOnz+VJx3/43o+P9UuC 9yAqsTKxO6ROsDeDzU686CFR+udHkm3JmrBO0sjq0B9gzRvdDQ9ec+I+m53viUEh 0Po=

=OFeM

-----END PGP PUBLIC KEY BLOCK-----

Если же экспортировать сертификат в файл с расширением. gpg или

.pgp, то экспорт осуществится в двоичном формате.

После этого файл сертификата можно передать любому желающему любым удобным способом (по электронной почте, через разделяемые файло- вые хранилища или через flash-накопители при личной встрече). Чтобы вос- пользоваться вашим открытым ключом, обладатель сертификата должен им- портировать его в свой менеджер ключей. Для этого используется команда File->Import Certificates (или одноименная кнопка на панели инструментов Cleopatra). Выбрав в открывшемся окне файл с сертификатом открытого ключа, пользователь импортирует его в систему (см сертификат с ID= B358DF69) на рис. 7.

Для того, чтобы убедиться в принадлежности ключа лицу, от имени которого фал сертификата был вам отправлен (например, по электронной почте), необходимо сверить его fingerprint (“отпечаток пальца”). Как уже было отмечено ранее, этот отпечаток (и даже его последние 8 цифр) уни- кально идентифицирует ключевую пару и ее владельца. Поэтому

достаточно в окне свойств импортированного ключа (открывается двойным щелчком по сертификату в окне рис.7) убедиться в соответствии отпечатка установленного ключа (рис.8) тому отпечатку, который заявляет истинный владелец ключевой пары и можно считать истинность ключа доказанной. Проблема получения информации о значении отпечатка ключевой пары от ее владельца может решаться различными способами: телефонный звонок, раз- мещение отпечатка владельцем на персональной Интернет-странице или ви- зитке. В любом случае, аутентичность этой информации должен проверять импортирующий.

Рисунок 8. Информация об импортированном сертификате

Специально для аутентификацию ключей пользователям GPG предусмотрен механзм установки доверия: пользователь, который убедился в истинности импортированного сертификата может сообщить остальным пользователям сстемы об этом, установив сертификату доверие от своего имени. Для сертификации стороннеого сертификата использлуется команда Certificates->Certify Certificate. В появившемся окне (рис.9а) необходимо выбрать, какой из установленных публичных ключей необходимо сертифицировать и указать, что отпечаток ключа был проверен. В следующем окне (9б) необходимо указать, от имени каокго пользователя (каким из зарегистрироанных закрытых ключей) будет сертифицироваться ключ. Далее следует стандартная защита от несанкционированного использования закрытого ключа – ввод парольной фразы (рис. 9в). Успешное завершение процесса сертификации сопровождается выводом сообветствуего окна (рис. 9г). После этого импортированный ключ можно увидеть в списке Trusted Certificates окна менеджера ключей.

а) Выбор ключа для установки доверия б) Выбираем, кто предоставляет доверие

в) Парольня фраза защищает закрытый г) Подтвержение предоставления доверия

ключ от несанкционированного использования

Рисунок 9. Процесс утановления доверия сертификату

Таким образом формируется сеть доверия «Web of Trust», в которой я доверяю некоторому ключу незнакомого мне пользлователя, если ему установил доверия другой пользлователь, которому я доверяю. Подобная цепочка доверия может быть и более длинной: «Я доверяю А, он доверяет В, В дверяет С, а, следовательно, я могу доверять С». Чтобы эта цепочка помогала принимать более ответственные решения о доверии, система позволяет определять степень доверия тому или иному ползователю: если в окне свойтсв сертифиакта (рис. 8) нажать кнопку «Trust Certifications Made by This Certificate…» («Доверять сертификациям, сделанным от имени данного сертификата»), то в открывшемся окне (рис.10) можно указать, насколько вы доверяете рекомендациям владельца этой ключевой пары.

Рисунок 10. Оценка доверия владельцу ключевой пары.

Возможны варианты:

I do not know

Я его не знаю

I do NOT trust them

Я му не доверяю

I believe checks are casual Частичное доверие владельцу сертификата

I believe checks are very accurate

Полное доверие владельцу сертификата

This is my certificate

Это мой сертификат

Неизвестный пользователь.

Игнорируется при проверке полинности сертификатов OpenPGP

Влалельцу ключа отказывается в доверии – он уличен в установке доверия непроверенным сертификатам

Владелец ключа старается сертифицировать сторонне ключи с проверкой, но не всегда. Если более 3-х пользователей установят такой уровень доверия сертификату, он принимается как заслуживающий доверия

Владелец ключа всегда строго проверяет сертифицируемые ключи. Если у ключа есть хотя бы один подобный уровень доверия, он считается заслужаваюим доверия

Этот уровень устанавливаюся для собственных сертификатов, они становятся доверенными

Установленный уровень доверия владельцу сертификата со стороны текущего пользователя указывается в окне свойств сертификата (рис. 6 под зеленым овалом).

Еще одной замечательной возможностью импорта/экспорта открытых ключей является сервер ключей. Выбрав пункт меню File->Export Certificates to Server…, можно опубликовать свой открытый ключ на общедоступном сервере ключей (предварительно адрес одного или нескольких серверов ключей должен ибыть добавлен в настройках Cleopatra). Пользователь может выбирать из множества проверенных серверов ключей OpenPGP, например:

hkp://keys.gnupg.net hkp://blackhole.pca.dfn.de hkp://pks.gpg.cz hkp://pgp.cns.ualberta.ca hkp://minsky.surfnet.nl hkp://keyserver.ubuntu.com hkp://keyserver.pramberger.at http://keyserver.pramberger.at http://gpg-keyserver.de

Открытый ключ, опубликованный на сервере, можно импоритировать в менеджер ключей, выбрав команду File->Lookup Certificates on Server. В прделагаемом окне (рис.11) предлагается найти сертификат по имени или почтовому адресу и импортировать, нажав соответствующую кнопку.

2.4. Шифрование и подпись документов.

Теперь, когда создана пара ключей и открытый ключ передан получателю сообщений, настало время приступить непосредственно к защите информации. Для этих целей подойдет как менеджер ключей Cleopatra, так и специальное рас- ширение для ОС Windows под названием GPG Explorer eXtension (GpgEX), кото- рое делает доступными команды шифрации/дешифрации и формирова- ния/верификации ЭЦП доступными из контекстного меню Windows. Для того, чтобы зашифровать файл (сформировать электронную подпись для него) выбира- ем команду File->Sign/Encrupt Files… в Cleopatra или пункт Подписать и зашиф- ровать… в контекстном меню шифруемого файла или папки в проводнике Win- dows (рис.12). контекстном меню про- водника

В ответ на команду откроется окно (рис.13а), в котором необходимо вы- брать требуемую операцию (только шифровать, только подписать, и то, и другое). Помимо непосредственно защиты система предлагает сжать файл с результатом работы, а также осуществить транспортное кодирование (перевод результата в поток ASCII-байт), а также удалить исходный документ (если выбран режим с шифрованием).

Если был выбран режим шифрования, в следующем окне (рис. 13б) система предложит выбрать сертификат(-ы) открытого (-ых) ключа (-ей), с помощью ко- торого(-ых) будет шифроваться документ. Можно выбрать несколько сертифика- тов, тогда сообщение смогут расшифровать несколько получателей – владельцев парных закрытых ключей. Рекомендуется (а если файл будет удаляться после шифрования, то настоятельно) включить в список сертификатов для шифрации и одни из своих сертификатов – иначе невозможно будет самому расшифровать до- кумент.

Если выбранный режим работы предполагает подпись документа, то откро- ется окно выбора сертификата закрытого ключа, которым будет подписан доку- мент. В выпадающем списке OpenPGP Signing Certificate этого окна будут ото- браны только сертификаты закрытых ключей, зарегистрированных в Cleopatra. Для защиты закрытого ключа от несанкционированного использования система предложит ввести парольную фразу, заданную при генерации ключевой пары (см. пункт 2.2). Удачное завершение процесса шифрации/подписи будет подтверждено выводом позитивного окна (рис. 13.д).

В результате выполнения операций шифрации/формирования подписи на диске в папке с документов появятся документы:

<имя\_файла>.gpg Файл с зашифрованным содержимым исходного файла

<имя\_файла>.gpg.asc Файл с зашифрованным содержимым

исходного файла (если в окне 13а был выбран режим for output as text/ASCII armor)

а) Выбор режима б) Выбор сертификата ключа шифрации

в) Выбор серификата ключа подписи г) Ввод парольной фразы

д) отчет об успешном выполнении операции Рисунок 13. Процесс шифрации/формирования подписи

<имя\_файла>.sig Файл с электронной подписью документа

<имя\_файла>.asc Файл с электронной подписью документа (если в окне 13а был выбран режим for output as text/ASCII armor)

Эти файлы можно передать получателю, чтобы он смог их расшифро- вать/проверить подпись под документом. Для этих целей получатель использует либо команду File->Decrypt/Verify Files… , либо команду Расшифровать/ Прове- рить… контекстного меню файла с шифром/подписью в проводнике Windows.

В открывшемся окне (рис. 14) можно уточнить параметры процесса дешиф- рации/проверки подписи: указать, какой файл был подписан, если происходит проверка электронной подписи (поле Input file is a detached signature), уточнить каким алгоритмом был дополнительно упакован файл в поле Input file is an archive, unpack with, а также задать папку, в которую будут помещены результаты выполнения операции в поле Output folder.

Рисунок 14. Окно настройка процесса дешифрации файла/верификации ЭЦП.

После нажатия кнопки Decrypt/Verify программа выполнит над выбранным файлом указанную операцию (в случае выполнения дешифрации предварительно запросит парольную фразу) и результат работы будет отображен в окне вида, представленного на рис. 15.

Если при проверке ЭЦП был использован верный открытый ключ и подпи- санный файл не изменялся с момента формирования подписи, то окно результата выполнения команды будет иметь вид как на рис.15а, если файл был изменен, то результатом операции будет окно вида 15б. Если при дешифрации будет трое- кратно введена некорректная парольная фраза – пользователь увидит окно 15в. Если же дешифрация будет произведена успешно – на диске в указанной в окне рис.14 папке будет размещен результат дешифрации.

а) ЭЦП верна б) ЭЦП неверна

в) Введена неверная парольная фраза

Рисунок 15. Виды окон с результатами проверки ЭЦП или дешифрации файла

Помимо шифрации файлов GPG поддерживает шифрование/дешифрацию, а также подпись/проверку ЭЦП для буфера обмена Windows (кнопка Clipboard на панели инструментов Cleopatra), а также почтовых сообщений в поддерживаемых почтовых клиентах (подробности см. в документации, поставляющейся вместе с системой).

Еще одной востребованной функцией GPG является формирова- ние/проверка контрольной суммы документов с использованием хеш-функции SHA.

Подав команду File->Create Checksum Files.., можно выбрать несколько файлов на диске, для которых будут вычислены криптостойкие контрольные сум- мы. Их значения будут занесены в файл sha1sum.txt в виде:

7b21848ac9af35be0ddb2d6b9fc3851934db8420 F:/gpg/важно.txt 7b21848ac9af35be0ddb2d6b9fc3851934db8420 F:/gpg/важно2.txt 7b21848ac9af35be0ddb2d6b9fc3851934db8420 F:/gpg/важно4.txt

где в каждой строке указывается хеш-дайджест файла, а также его имя и путь к нему.

Впоследствии можно проверить целостность файлов, сверив их текущие контрольные суммы с записанными в файле. Выполнить эту операцию можно с помощью команды File->Verify Checksum Files.. Результаты проверки будут крас- норечиво отображены в окне отчета (рис. 16).

Как и для прочих уже упоминавшихся команд, для команд формирова- ния/проверки контрольной суммы есть альтернативная форма инициализации че- рез контекстное меню файлов (команды Создать контрольные суммы/Проверить контрольные суммы) в подменю Другие параметры GpgEX.

2.4. Интерфейс командной строки.

Функции GnuPG можно использовать не только в графическом интерфейсе. Как всякая система, зародившаяся в среде Unix, GnuPG позволяет выполнять все вышеперечисленные действия по консольным командам. Такой интерфейс работы с программой, возможно, не настолько нагляден и прост для обычного пользова- теля, но он позволяет автоматизировать и масштабировать криптографические процедуры, включая их в состав командных файлов или исполняя по расписанию планировщика задач.

Для использования возможностей системы GnuPG в консольном режиме предназначена внешняя команда gpg. Ее можно вызвать с параметрами как в ко- мандной строке, так и в составе командных файлов. Команда gpg поддерживает все вышеперечисленные режимы работы системы: генерацию, экспорт и импорт ключей, шифрование и дешифрацию файлов, формирование и проверку элек- тронной подписи. Для gpg актуально соглашение о статусе завершения – если ко- манда завершилась с кодом 0, то команда выполнилась успешно, код завершения 1 и выше соответствует ошибке, возникшей при выполнении команды.

Конкретное действие, выполняемое командой gpg зависит от параметров запуска. Рассмотрим основные:

-s, –sign

Создать подпись для документа. Команда может быть подана вместе с па- раметром –encrypt.

-b, –detach-sign

Создать отдельный файл с подписью.

-e, –encrypt

Зашифровать данные. Эта опция может быть подана вместе с –sign.

-c, –symmetric

Зашифровать только с использованием симметричного шифра.

–decrypt <имя\_файла>

Дешифровать файл (или стандартный поток ввода, если файл не задан) и вывести результат в стандартный поток вывода (или в файл, заданный в оп- ции –output). Если файл был дополнительно подписан, будет проверена корректность электронной подписи.

–verify <имя\_файла>

Проверка подписи под файлом как внедренной в файл (расширение gpg),

так и в отдельном файле (расширение sig)

–list-keys [<имена>]

Вывод списка всех зарегистрированных открытых ключей (всех или тех, ко- торым соответствует заданное имя).

–list-secret-keys [<имена>]

Получить список установленных закрытых ключей (всех или тех, которым соответствует заданное имя).

–fingerprint [<имена>]

Вывести список всех ключей вместе с их отпечатками.

–gen-key

Создать новую пару ключей.

–edit-key <ID ключа>

Открывает меню с возможностью выполнения операций с ключом: смена имени пользователя, парольной фразы, установление доверия ключу, запре- тить/разрешить ключ и др.

–delete-key имя

Удалить публичный ключ

–delete-secret-key имя

Удалить ключевую пару

–export <имя>

Экспортировать все ключи или ключ с заданным именем. Имя файла экс- порта можно задать с помощью опции --output.

–import <имя\_файла>

Импортировать ключ из указанного файла.

Помимо опций, задающих характер выполняемых действий, команде gpg можно задать ряд конфигурационных опций, рассмотрим назначение некоторых из них (полный список можно посмотреть в [5])

-a, –armor

Выводить информацию в ASCII-формате

-o, –output <файл>

Выводить информацию в указанный файл.

-u, –local-user <имя>

Использовать ключ пользователя с указанным именем для формирования подписи

–default-key <имя>

Использовать указанного пользователя по умолчанию при формировании подписи.

–keyserver <имя>

Использовать указанный сервер ключей, если ключ не найдется среди за- регистрированных.

–cipher-algo <название>

Использовать указанный алгоритм шифрования. Список доступных алго- ритмов можно получить, используя опцию –version.

–digest-algo <название>

Использовать указанный алгоритм формирования подписи. Список доступ- ных алгоритмов можно получить, используя опцию –version.

–throw-keyid

Не размещать информацию об используемом ключе в зашифрованное со- общение. Повышает стойкость шифротекста к криптоанализу, но замедляет дешифрацию, т.к. приходится перебирать все доступные ключи

–s2k-cipher-algo <название>

Использовать указанный шифр для защиты закрытого ключа. По умолча- нию используется Blowfish

Примеры выполнения криптографических операций с использованием консольной команды gpg:

Зашифровать файл публичным ключом указанного пользователя

gpg –u <ID пользователя> –е <имя\_исходного\_файла>

Зашифровать файл симметричным алгоритмом (ключ будет запрошен)

gpg –-output <имя\_файла\_с\_шифром> –c <имя\_исходного\_файла>

Расшифровать зашифрованное сообщение в указанный файл

gpg –output <имя\_файла> --decrypt <зашифрованный\_файл>

Подписать файл от имени пользователя

gpg –u <ID пользователя> --sign <подписываемый\_файл>

Проверить электронную подпись под файлом (подпись будет размещена в отдельном файле с расширением asc)

gpg –u <ID пользователя> --detach-sign

<подписываемый\_файл>

Проверить электронную подпись под файлом

gpg –verify <имя\_файла\_подписи>

2.5. Порядок выполнения лабораторной работы.

Практическое занятие выполняется студентами в паре для полноценного об- мена ключами, зашифрованными и подписанными сообщениями. Каждый из сту- дентов в паре работает на компьютере с установленной системой GnuPG для Win- dows. Компьютеры должны быть объединены в сеть для оперативного обмена файлами. Порядок работы каждого из студентов в паре:

2.5.1. Создайте пару ключей в менеджере ключей Cleopatra.

2.5.2. Скопируйте произвольный текст в буфер обмена. Зашифруйте содержи- мое буфера обмена с помощью своего открытого ключа. Вставьте содержимое буфера обмена в текстовый редактор, убедитесь, что оно зашифровано. Те- перь скопируйте в буфер шифротекст, дешифруйте его своим закрытым клю- чом, вновь вставьте содержимое буфера обмена в текстовый редактор, убеди- тесь, что текст был успешно расшифрован.

2.5.2. Экспортируйте сертификат открытого ключа из своей пары ключей в файл и передайте его своему напарнику.

2.5.3. Получив файл с экспортированным ключом от напарника, импортируйте его в менеджер ключей. Установите для импортированного ключа полное до- верие.

2.5.4. Зашифруйте с использованием импортированного ключа напарника произвольный текст на диске. Передайте зашифрованный текст напарнику.

2.5.5. Получив зашифрованный файл от напарника, дешифруйте его своим за- крытым ключом. Убедитесь, что файл был успешно дешифрован.

2.5.6. Используя свой закрытый ключ, подпишите произвольный файл на дис- ке электронной подписью. Передайте подписанный документ вместе с подпи- сью напарнику.

2.5.7. Получив от напарника документ с подписью, убедитесь, что подпись верна. Измените подписанный документ и убедитесь, что подпись стала не- верна. Верните документ к первоначальному состоянию и вновь убедитесь, что подпись верна.

2.5.8. Скопируйте во временную папку несколько документов. Сформируйте для этих документов файл с контрольными суммами. Внесите изменения в один или несколько документов и убедитесь, что система обнаружит расхож- дения контрольных сумм.

2.5.9. Выполните индивидуальное задание в соответствии с заданием. Задание предполагает написание командного скрипта, выполняющего преобразования данных в соответствии с заданием

2.5.9.1. Написать скрипт, который с ключом /e будет шифровать все тексто- вые документы в заданной параметром скрипта папке, а с ключом /d дешиф- ровывать их.

2.5.9.2. Написать скрипт, который будет с использованием ЭЦП контроли- ровать целостность файлов в заданной папке.

2.5.9.3. Написать скрипт, который будет с ключом /s сканировать заданную папку и для всех новых файлов в ней создавать электронную подпись, а с

ключом /v проверять подписи под файлами и формировать отчет о корректно- сти подписей в файле-отчете.

2.5.9.4. Написать скрипт, который с будет формировать/проверять общую подпись для всех текстовых файлов (как единого целого) в заданной папке.

2.5.9.5. Написать скрипт, который будет шифровать и подписывать тексто- вые файлы в заданной папке с ключом /e, а дешифровывать при запуске с ключом /d только те файлы, которые прошли проверку на корректность элек- тронной подписи.

2.5.9.6. Написать скрипт, который будет шифровать и дешифровывать фай- лы в папке указанной в качестве параметра скрипта, от имени пользователя, идентификатор которого также передается в скрипт как параметр.

2.5.6.7. Написать скрипт, который будет выполнять сое содержимое только после проверки собственной целостности с помощью электронной подписи.

2.5.10. Составьте отчет по лабораторной работе

2.5.11. Ответьте на контрольные вопросы к лабораторной работе. Отчитайте лабораторную работу преподавателю.

2.6. Контрольные вопросы

1. Перечислите достоинства и недостатки асимметричных алгоритмов в срав- нении с симметричными.

2. Перечислите основные трудновычислимые задачи, использующиеся в со- временной асимметричной криптографии.

3. Какую роль в современных криптосистемах играют симметричные и асим- метричные алгоритмы шифрования?

4. Как обеспечивается доверие к открытым ключам асимметричных шифров и верификации ЭЦП? Как решается этот вопрос в системе GPG?

5. Какую роль в стандарте OpenPGP играют сервера ключей? Как их можно использовать в системе GPG?

6. Какие стандарты поддерживает система GPG?

7. Какая схема сертификации ключей реализована в системе GPG? Перечисли- те ее достоинства и недостатки.

8. Опишите схему работы асимметричной криптосистемы.

9. Опишите схему формирования-верификации ЭЦП.

10. Объясните, чем отличается процесс проверки контрольных сумм и ЭЦП в системе GPG. Какой из методов контроля надежнее?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 24 Аутентификация пользователей Web-систем средствами технологии РНР**

1. ЦЕЛЬ РАБОТЫ: изучение принципов аутентификации пользова- телей в Web-системах на примере РНР-сеансов.

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить принципы аутентификации пользователей в Web-

системах.

2. Реализовать систему аутентификации с помощью РНР-сеансов.

3. СОДЕРЖАНИЕ ОТЧЕТА

1) титульный лист;

2) формулировку цели работы;

3) описание результатов выполнения;

4) выводы, согласованные с целью работы.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 25Защита информации с помощью пароля

1 ЦЕЛЬ РАБОТЫ

Целью работы является исследование защиты с применением пароля, а также исследование методов противодействия атакам на пароль.

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТ

2.1. Проведение атаки перебором (bruteforce attack)

1. Используя программу для вскрытия паролей произвести атаку на зашифрованный файл try\_me.rar (try\_me.arj, try\_me.zip – в зависимости от варианта). Область перебора – все печатаемые символы, длина пароля от 1 до 4 символов. Время выполнения на компьютере класса Pentium примерно 3-4 минуты. На компьютере класса Pentium II – 50 секунд. Проверить правильность определенного пароля, распаковав файл и ознакомившись с его содержимым.

2. Выполнив пункт 1, сократить область перебора до фактически используемого (например если пароль 6D1A – то выбрать прописные английские буквы и цифры). Провести повторное вскрытие. Сравнить затраченное время.

2.2. Проведение атаки по словарю (dictionary attack)

1. Сжать какой-либо небольшой файл, выбрав в качестве пароля английское слово длиной до 5 символов (например love, god, table, admin и т.д.). Провести атаку по словарю. Для этого выбрать вид атаки и в закладке Словарь выбрать файл English.dic. Он содержит набор английских слов и наборы символов, наиболее часто использующиеся в качестве паролей.

2. Попытаться определить пароль методом прямого перебора. Сравнить затраченное время.

3. СОДЕРЖАНИЕ ОТЧЕТА

1) титульный лист;

2) формулировку цели работы;

3) описание результатов выполнения;

4) выводы, согласованные с целью работы.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие виды атак на пароль Вы знаете?

2. Что такое плохой пароль?

3. Как можно противостоять атаке полным перебором?

4. Как длина пароля влияет на вероятность раскрытия пароля?

5. Какие рекомендации по составлению паролей Вы можете дать?

**ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №26**

1. ЦЕЛЬ РАБОТЫ

1.1. Ознакомиться с основными методами криптографической защиты информации.

1.2. Получить практические навыки создания ПО по криптографическим преобразованиям информации.

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

На языке C++ или Pascal написать программу шифрования и дешифрования текстового файла методом, указанным преподавателем.

3. СОДЕРЖАНИЕ ОТЧЕТА

3.1. Название работы.

3.2. Цель работы.

3.3. Тексты программ.

3.4. Общие выводы, сделанные в процессе выполнения лабораторной работы.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ:

4.1. Цель и задачи криптографии.

4.2. Симметричные криптосистемы: шифры перестановки.

4.3. Симметричные криптосистемы: шифры простой замены.

4.4. Симметричные криптосистемы: шифры сложной замены.

4.5. Симметричные криптосистемы: гаммирование.

4.6. Асимметричные криптосистемы, схема шифрования Эль Гамаля.

# Практическое занятие №27 Удостоверяющие центры на основе службы сертификации в операционной системе Windows 2003 Server

##### Введение

Работа выполняется в виртуальной машине. Операционная систем Windows 2003 Server. Пароль администратора пустой.

Запуск Виртуальной машины:

 Запуск консоли виртуальной машины. Пуск => Все программы => Microsoft Virtual PC



 Удалить из консоли виртуальной машины всю информацию

 Добавить на консоль информацию по виртуальной машине Windows 2003 Server

4. Запустить виртуальную машину с ОС Windows 2003 Server

1. Установка центра Сертификации.
2. В мастере компонентов Windows добавить компонент Обновление корневых сертификатов и Службы сертификации

1. В ходе выполнения установки Службы сертификации выбрать “Изолированный корневой ЦС”

1. Задать имя Центра сертификации. (XXXXXX – заменить )
2. Входе установки будет запрошен диск дистрибутива Windows. Указать путь: C:\Distrb\Win2003Sp1\I386
3. Посмотреть результаты установки. Для этого:
	1. запустить консоль управления - Пуск => Выполнить => mmc



* 1. В открывшейся консоли добавить оснастку – Центр сертификации



* 1. Ознакомиться с содержимым созданного центра сертификации. Просмотреть сертификат центра и все папки. Скопировать сертификат центра в файл.



## Настройка КриптоПро CSP

* 1. Из панели управления запустить КриптоПро CSP
	2. Перейти на вкладку Оборудование
	3. Настроить считыватели, выбрав Дисковод и Реестр



* 1. Настроить датчик случайных чисел, выбрав Биологический ДСЧ



* 1. На этом настройка КриптоПро CSP завершена

## Запрос сертификата. Обработка запроса.

* 1. На рабочем столе открыть ярлык запроса сертификата.



* 1. Открыть ярлык расширенный запрос сертификата



* 1. Открыть ссылку Создать и выдать запрос к этому ЦС.
	2. Заполнить необходимые поля запроса и выдать запрос на сертификат.
	3. Обработка запроса на сертификат
		1. В консоли управлении Вашего Центр сертификации перейти на вкладку “Запросы в ожидании”

ii. Обработать существующий запрос.

iii. Перейти на вкладку “Выданные сертификаты”. Открыть выданный сертификат и ознакомиться с его содержимым

* 1. Получение сертификата.
		1. Подключиться к WEB серверу Центра сертификации по ссылке

##### <http://localhost/certsrv/>

ii. Выберите действие Просмотр состояния ожидаемого запроса сертификата

iii. Выбрать сертификат, который вы хотите получить и установите его

1. Отмена настроек КриптоПро CSP. Отменить все настройки введенные Вами
2. Удаление центра Сертификации.. Удалить Центр Сертификации

## Введение

Пиратство в сфере информационных технологий и в части нелегального использования чужого программного обеспечения, становится доходным и динамично развивающимся полулегальным бизнесом с такими естественными составляющими, как производство, логистика, дистрибуция, прямые продажи и пр. Приобретение и использование контрафактной (т. е. незаконной, ворованной) продукции, и тем более кража программных разработок и нажива на них, — это серьёзные преступления, которые наносят гигантский ущерб правообладателям. В связи с этим всё более актуальными становятся технологические вопросы защиты ПО.

Вопрос о защите программного обеспечения появился практически тогда же, когда появилось и оно само, и приобрёл особый вес с распространением персональных компьютеров. Всеобщая компьютеризация, помимо безусловных плюсов, таких как новые отрасли экономики, новые рынки, доступность богатейших библиотек, возможность общения, самореализации т. д., породила и ряд очевидных побочных эффектов, одним из которых является теневой бизнес по изготовлению и продаже нелицензионного программного обеспечения. Незаконно растиражированные программы свободно продаются на рынках, и, покупая всё это, пользователь поощряет процветание и развитие пиратского бизнеса. Чего же лишается потребитель, покупая «нелицензионки»? Гарантий качества продукта, технической документации и поддержки, бесплатных или льготных обновлений более совершенных версий ПО и ряда внешних атрибутов (фирменного оформления, вкладок и пр.). Впрочем, последнее, естественно, не может сформировать комплекс вины у обладателя пиратского диска.

Что на этом фоне остаётся делать программистам? Позаботиться о защите собственноручно написанных программ самим, не дожидаясь, пока представителей нелегального бизнеса вдруг заест совесть и они выйдут из тени.

На сегодняшний день есть несколько способов защиты своих программных продуктов от пиратов. Подавляющее большинство создателей ПО используют различные программные модули, контролирующие доступ пользователей с помощью ключей активации, серийных номеров и т. д. Но невысокая стоимость — это, наверное, единственный плюс таких решений для борьбы со взломом программ. Интернет изобилует разного рода «крэками»1, патчами и эмуляторами, позволяющими обходить защиту и блокирующими запрос паролей или ключей активации. О серийных ключах говорить и вовсе не приходится. Легальный пользователь, купивший лицензионную программу, может просто обнародовать код, по каким-то своим личным весьма альтруистическим соображениям, и не пройдёт и пары часов, как им воспользуется не одна сотня приверженцев всего бесплатного.

Эти очевидные недостатки привели к созданию аппаратной защиты программного обеспечения в виде электронного ключа. Первый ключ для защиты ПО на персональном компьютере был создан в начале 1980-х годов и с тех пор претерпел значительные изменения. Сегодня он представляет собой компактное USB- устройство (менее 4 см в длину), позволяющее шифровать данные, используя либо публичный, либо секретный алгоритмы.

Секретные алгоритмы разрабатываются самим производителем средств защиты, в том числе и индивидуально для каждого заказчика. Главным недостатком использования таких алгоритмов является невозможность оценки стойкости. С уверенностью сказать, насколько надёжен алгоритм, можно было лишь постфактум: взломали или нет. Кроме того, знание алгоритма даёт возможность создания эмулятора — программного продукта, полностью выполняющего все функции аппаратного устройства.

Публичный алгоритм, или «открытый исходник», обладает криптостойкостью несравнимо большей. Брюс Шнайер2 в своей статье «Открытые исходники и безопасность» объясняет преимущества публичных алгоритмов прежде всего тем, что они проверены не случайными людьми, а рядом экспертов, специализирующихся на анализе криптографии. «Прежде чем алгоритм будет признан надёжным, он должен быть изучен многими экспертами в течение ряда лет. И это — сильный довод в пользу криптографии с открытым кодом.

Поскольку единственный способ обрести уверенность в алгоритме — это предоставить его для изучения экспертам, а единственное условие, при котором они будут тратить время на проверку, — возможность публиковать результаты своих исследований, алгоритмы должны публиковаться», — пишет автор. Таким образом, становится очевидным, что «открытые исходники» разрабатываются с учётом того, что будут доступны всем же-лающим. Следовательно, риска их публикации нет.

Публичный криптоалгоритм кардинально усложняет задачу взлома ПО, сводя её к криптоанализу и вычислению ключей шифрования методом полного перебора. Самыми известными открытыми алгоритмами на сегодняшний день являются DES, Triple DES, Blowfish, RC2, CAST и, наверное, один из самых «продвинутых»

на сегодняшний день, общепризнанный американский стандарт шифрования AES (Advanced Encryption Standard), пришедший на смену двум первым.

## USB-ключи для защиты программ и данных от копирования, нелегального использования и несанкционированного распространения

1. Проблема защиты программ

Пиратство в сфере информационных технологий и в части нелегального использования чужого программного обеспечения, становится доходным и динамично развивающимся полулегальным бизнесом с такими естественными составляющими, как производство, логистика, дистрибуция, прямые продажи и пр. Приобретение и использование контрафактной (т. е. незаконной, ворованной) продукции, и тем более кража программных разработок и нажива на них, — это серьёзные преступления, которые наносят гигантский ущерб правообладателям. В связи с этим всё более актуальными становятся технологические вопросы защиты ПО.

Вопрос о защите программного обеспечения появился практически тогда же, когда появилось и оно само, и приобрёл особый вес с распространением персональных компьютеров. Всеобщая компьютеризация, помимо безусловных плюсов, таких как новые отрасли экономики, новые рынки, доступность богатейших библиотек, возможность общения, самореализации т. д., породила и ряд очевидных побочных эффектов, одним из которых является теневой бизнес по изготовлению и продаже нелицензионного программного обеспечения. Незаконно растиражированные программы свободно продаются на рынках, и, покупая всё это, пользователь поощряет процветание и развитие пиратского бизнеса. Чего же лишается потребитель, покупая «нелицензионки»? Гарантий качества продукта, технической документации и поддержки, бесплатных или льготных обновлений более совершенных версий ПО и ряда внешних атрибутов (фирменного оформления, вкладок и пр.). Впрочем, последнее, естественно, не может сформировать комплекс вины у обладателя пиратского диска.

## Пути и методы решения проблемы

Что на этом фоне остаётся делать программистам? Позаботиться о защите собственноручно написанных программ самим, не дожидаясь, пока представителей нелегального бизнеса вдруг заест совесть и они выйдут из тени.

На сегодняшний день есть несколько способов защиты своих программных продуктов от пиратов. Подавляющее большинство создателей ПО используют различные программные модули, контролирующие доступ пользователей с помощью ключей активации, серийных номеров и т. д. Но невысокая стоимость — это, наверное, единственный плюс таких решений для борьбы со взломом программ. Интернет изобилует разного рода «крэками»1, патчами и эмуляторами, позволяющими обходить защиту и блокирующими запрос паролей или ключей активации. О серийных ключах говорить и вовсе не приходится. Легальный пользователь, купивший лицензионную программу, может просто обнародовать код, по каким-то своим личным весьма альтруистическим соображениям, и не пройдёт и пары часов, как им воспользуется не одна сотня приверженцев всего бесплатного.

Эти очевидные недостатки привели к созданию аппаратной защиты программного обеспечения в виде электронного ключа. Первый ключ для защиты ПО на персональном компьютере был создан в начале 1980-х годов и с тех пор претерпел значительные изменения. Сегодня он представляет собой компактное USB- устройство (менее 4 см в длину), позволяющее шифровать данные, используя либо публичный, либо секретный алгоритмы.

Секретные алгоритмы разрабатываются самим производителем средств защиты, в том числе и индивидуально для каждого заказчика. Главным недостатком использования таких алгоритмов является невозможность оценки стойкости. С уверенностью сказать, насколько надёжен алгоритм, можно было лишь постфактум: взломали или нет. Кроме того, знание алгоритма даёт возможность создания эмулятора — программного продукта, полностью выполняющего все функции аппаратного устройства.

Публичный алгоритм, или «открытый исходник», обладает криптостойкостью несравнимо большей. Брюс Шнайер2 в своей статье «Открытые исходники и безопасность» объясняет преимущества публичных алгоритмов прежде всего тем, что они проверены не случайными людьми, а рядом экспертов, специализирующихся на анализе криптографии. «Прежде чем алгоритм будет признан надёжным, он должен быть изучен многими экспертами в течение ряда лет. И это — сильный довод в пользу криптографии с открытым кодом.

Поскольку единственный способ обрести уверенность в алгоритме — это предоставить его для изучения экспертам, а единственное условие, при котором они будут тратить время на проверку, — возможность публиковать результаты своих исследований, алгоритмы должны публиковаться», — пишет автор. Таким образом, становится очевидным, что «открытые исходники» разрабатываются с учётом того, что будут доступны всем же-лающим. Следовательно, риска их публикации нет.

Публичный криптоалгоритм кардинально усложняет задачу взлома ПО, сводя её к криптоанализу и вычислению ключей шифрования методом полного перебора. Самыми известными открытыми алгоритмами на сегодняшний день являются DES, Triple DES, Blowfish, RC2, CAST и, наверное, один из самых «продвинутых» на сегодняшний день, общепризнанный американский стандарт шифрования AES (Advanced Encryption Standard), пришедший на смену двум первым.

## Практическое занятие № 28. Защита программ от несанкционированного использования с помощью USB-ключей и программного обеспечения производителя.

1. Цель работы

Ознакомиться с устройством USB ключей HASP. Программное обеспечение, поставляемое с устройством. Научиться защищать программы от не санкционированного использования с помощью программного обеспечения производителя ключей HASP

## Краткие теоретические сведения

Для эффективного использования системы защиты HASP Вам следует ознакомиться с принципами её работы и терминологией, изложенными в настоящей главе. Если Вы собираетесь использовать ключ NetHASP, Вам следует также прочитать Главу 4 для ознакомления с принципами работы и терминологией NetHASP.

##### Ключи HASP

Ключи HASP бывают следующих типов: Локальные ключи

Локальные ключи – это ключи HASP, предназначенные для автономных (не являющихся частью сети) компьютеров. К этой категории относятся все ключи, кроме NetHASP. Демонстрационные ключи. В каждый Комплект разработчика HASP входит демонстрационный ключ HASP (также демо-ключ). Демо-ключи обладают всеми возможностями ключей своего класса, но имеют стандартный демонстрационный код разработчика (см. стр. 38). Использовать демо-ключи для защиты программного обеспечения нельзя, поскольку они доступны для любого человека. Эти ключи лучше всего использовать для оценки системы защиты HASP.

##### Память HASP

Все ключи HASP, за исключением HASP4 без памяти, имеют перезаписываемую память. Используя память HASP, Вы можете делать следующее: Управлять доступом к различным программным модулям и пакетам программ. Назначить каждому пользователю Ваших программ уникальный номер. Сдавать программы в аренду и распространять их демо-версии с ограничением количества запусков. Хранить в ключе пароли, фрагменты кода программы, значения переменных и другую важную информацию.

Тип ключа Размер памяти HASP4 без памяти Нет HASP4 М1 112 байт

HASP4 М4 496 байт

HASP4 Time 512 байт

Все ключи NetHASP 496 байт

Идентификатор HASP

У каждого ключа HASP с памятью имеется уникальный опознавательный номер (ID- номер), или идентификатор, доступный для контроля защищёнными приложениями. Идентификаторы позволяют Вам различать пользователей приложений. Проверяя в программе идентификатор HASP, Вы можете предпринимать те или иные действия в зависимости от наличия конкретного ключа. Вы не можете заказывать ключи HASP с заранее заданными идентификаторами. Они назначаются псевдослучайным образом в процессе изготовления ключей, чем гарантируется защита от повтора.

##### Способы защиты HASP

Система HASP позволяет защищать программное обеспечение двумя различными способами: Утилитой HASP Envelope (оболочка) HASP API (Application Programming Interface – программный интерфейс приложения) Оболочка HASP

Использование HASP Envelope является основным способом защиты. Исполняемый файл заключается в защитную программную оболочку, кодирующую файл, и обладающую такими свойствами, как распознавание ключа и антиотладка. Оболочка не позволяет файлу выполняться без соответствующего ключа HASP.

Защита оболочкой производится быстро и без особых усилий. В то же время, она достаточно надёжна, так как делает отладку и дизассемблирование Ваших программ практически невозможными. Для защиты оболочкой исходные тексты программ не требуются.

Программный интерфейс пользователя HASP (API) Если у Вас имеются исходные тексты программы, которую надо защитить, то Вы можете пристыковать к ней модуль HASP API – объектный файл или библиотеку DLL. Поскольку модуль API сам по себе защищён и зашифрован, этот метод обеспечивает высокую степень защиты. API позволяет обращаться к ключу

## Порядок выполнения работы

#####  Установка программного обеспечения HASP

1. В списке оборудования компьютера убедиться в отсутствии устройств компании Aladdin (eToken, HASP). Если эти устройства присутствуют в списке оборудования – удалить их.
2. Удалить, если присутствует, программное обеспечение eToken и HASP.
3. Подключиться к компьютеру Centurion, открыть ресурс Q и войти в каталог setup.
4. Запустить программу setup.exe.
5. Следуя указаниям программы установки установить на компьютере программное обеспечение системы HASP. Установить компоненты помеченные ниже.



1. Подключить HASP к разъему USB компьютера.
	1. В случае правильной установки в электронном ключе HASP должен загореться световой индикатор.
	2. Если индикатор не загорается, обновить драйвер устройства вручную.
2. В списке оборудования компьютера найти подключенное устройство.

.

#####  Работа с программным обеспечением HASP

1. Запустить программу Aladdin DiagnostiX

Провести тестирование электронного ключа HASP.

 

1. Запустить демонстрационную программу – HASP Demo for Win16 . Описать работу этой программы
2. Запустить демонстрационную программу – HASP Demo for Win32 . Описать работу этой программы

 Зашита приложений с помощью HASP.

Используя инструмент HASP Envelope защитить избранное приложение от не санкционированного запуска. Санкционированный запуск – при наличии HASP ключа в системе.

Неанкционированный запуск – при отсутствии HASP ключа в системе.

1. Создать папку: Защищенные приложения
2. Скопировать в нее 3 исполняемых файла, например программу калькулятор – calc.exe
3. С помощью HASP Envelope защитить выбранные программы от несанкционированного запуска.
4. Запустить программу.
5. Закрыть программу
6. Вытащить HASP из разъема USB
7. Запустить защищенную программу.

Попробовать варианты защиты нескольких программ с одним ключом HASP

## Содержание отчета

Форма отчётности - подробное описания проделанной работы по каждому шагу в электронном документе (Microsoft Word). Привести структуру ключевой дискеты. Содержание файла открытых и закрытых ключей.

## Контрольные вопросы

1. Рассказать об утилите HASP Envelope
2. Рассказать об утилите тестирования HASP
3. Рассказать об утилите HASPEdit
4. Что такой идентификатор HASP
5. Как шифруются и дешифруются данные для распознавания ключа HASP

## Практическое занятие № 29. Защита программ от несанкционированного использования с помощью USB-ключей и средств разработчика.

* 1. Цель работы

Ознакомиться с устройством USB ключей HASP. Программное обеспечение, поставляемое с устройством. Научиться защищать программы от не санкционированного использования с помощью средств разработчика

## Краткие теоретические сведения

##### Информация для разработчика

Заказывая ключи у АЛАДДИНа, Вы получите ключи, содержащие информацию, специфическую для Вашей фирмы. Эта информация служит для отличия Ваших ключей от ключей других разработчиков.

Код разработчика

Код разработчика – это уникальный код, присваиваемый фирмой АЛАДДИН каждому разработчику программного обеспечения. Код разработчика «зашивается» в микросхему ASIC при изготовлении ключа и не поддаётся изменению, обеспечивая, таким образом, полную защиту от подделки.

Вам потребуется знание Вашего кода разработчика при заказе дополнительных ключей HASP. Код представляет собой комбинацию из 5 – 7 букв и цифр, напечатанных на этикетке каждого ключа. Пароли

Пароли HASP представляют из себя два целых числа, присвоенных каждому разработчику. Пароли однозначно связаны с кодом разработчика. Храните их в надежном месте, так как они необходимы Вам для доступа к ключу, защиты Ваших программ и использования входящих в комплект утилит.

##### Контроль наличия ключа

Осуществляя защиту приложения, Вы постоянно контролируете наличие ключа. Система HASP реализует этот контроль с помощью шифрования и дешифрования данных самим ключом.

Дешифрование данных для распознавания ключа. Распознавание наличия ключа HASP основано на использовании функций шифрования и дешифрования, что влечет за собой необходимость в некоторых действиях. Чтобы начать, Вам необходимо иметь уже некоторые данные, которые Вы ранее зашифровали. После этого Вы посылаете данные на ключ, используя функцию DecodeData. Происходит их дешифрование, в результате чего Вы можете проверить, верны ли дешифрованные данные. Если так, Вы можете сделать заключение о наличии ключа. Дешифрованные данные могут быть верифицированы как путем простого сравнения, так и более безопасным образом – использованием этих данных в Вашем защищенном приложении. Зашифрованные данные представляют собой функцию посланных на ключ данных и уникального, присвоенного разработчику, «кода разработчика». Вследствие этого, при шифровании одной и той же строки для двух разных разработчиков будет получен различный результат.

##### Память HASP

Все ключи HASP, за исключением HASP4 без памяти, имеют перезаписываемую память. Используя память HASP, Вы можете делать следующее: Управлять доступом к различным программным модулям и пакетам программ. Назначить каждому пользователю Ваших программ уникальный номер. Сдавать программы в аренду и распространять их демо-версии с ограничением количества запусков. Хранить в ключе пароли

## Порядок выполнения работы

Описание программы

### Программа Da Vinci Code была написана в учебных целях. В программе были использованы функции шифрования и дешифрования файлов средствами ключей HASP.

Данная программа может функционировать только под Windows- системами. В программе реализованы следующие функции распознание ключа, проверка пароля, шифрование и дешифрование файлов. Шифровании и дешифрование реализованы путем обращения к соответствующим функциям ключа. Для проверки ключа к файлу прикрепляется идентификатор ключа HASP.

Программой используется библиотека функций haspms32.dll.

Взаимодействие модуля Dll с основной программой происходит с помощью Windows сообщений.

Руководство пользователя:

Окно программы Da Vinci Code:



Рисунок 1 – Рабочее окно авторизации ключа.

Для авторизации ключа необходимо ввести пароли. В случае если пароли не совпадут будет выведено окно ошибки.



Рисунок 2 – окно ошибки авторизации.

В случае отсутствии ключа программа выдаст следующее окно.



Рисунок 3 – окно ошибки отсутствия ключа.

После ввода правильного пароля будет выведено основное око программы.



Рисунок 4 – Основное окно программы. Шифрование.

* Шифрование – в данной закладке происходит шифрование файлов;

### Зашифровать - при нажатии этой кнопки происходит процесс шифрования.

* + Исходный файл – в данном поле мы указываем файл который нам необходимо зашифровать;
	+ Зашифрованный файл – в данном поле указывается путь и название зашифрованного файла;



### Рисунок 5 – Окно результата шифрования.

* 1. Содержание отчета

Форма отчётности - подробное описания проделанной работы по каждому шагу в электронном документе (Microsoft Word). Привести структуру ключевой дискеты. Содержание файла открытых и закрытых ключей.

## Контрольные вопросы

* 1. Рассказать об утилите HASP Envelope
	2. Рассказать об утилите тестирования HASP
	3. Рассказать об утилите HASPEdit
	4. Что такой идентификатор HASP
	5. Как шифруются и дешифруются данные для распознавания ключа HASP

## Практическое занятие № 31. Защита данных с помощью USB-ключей и программного обеспечения производителя.

* + 1. Цель работы
		2. Краткие теоретические сведения
		3. Порядок выполнения работы
		4. Содержание отчета
		5. Контрольные вопросы

Практическое занятие № 30. Защита данных с помощью USB-ключей и средств разработчика.

1. Цель работы
2. Краткие теоретические сведения
3. Порядок выполнения работы
4. Содержание отчета
5. Контрольные вопросы

##### Введение

1. Программно-аппаратные средства криптографической защиты информации серии Криптон

Устройства КРИПТОН разработаны, производятся и реализуются Фирмой АНКАД. Они построены на разработанных Фирмой АНКАД специализированных 32-разрядных шифрпроцессорах серии БЛЮМИНГ. За 10 лет работы Фирма АНКАД поставила более 15 тысяч устройств КРИПТОН заказчикам в Центральном Банке, Федеральном агентстве правительственной связи и информации при Президенте РФ, министерствах обороны и внутренних дел, Министерстве по налогам и сборам, Федеральном казначействе, коммерческих банках, финансовых и страховых компаниях, многим корпоративным клиентам.

Сеть кооперационного производства устройств КРИПТОН охватывает наиболее известные предприятия российской электроники (ОАО “Ангстрем” и др.).

Устройства серии КРИПТОН имеют сертификаты соответствия требованиям ФАПСИ (в том числе в составе абонентских пунктов и автоматизированных рабочих мест для защиты информации, содержащей сведения, составляющие государственную тайну).

##### Концептуальный подход к защите информации в компьютерных системах

* 1. Устройства криптографической защиты данных серии криптон

Устройства криптографической защиты данных (УКЗД) серии КРИПТОН — это аппаратные шифраторы для IBM PC-совместимых компьютеров. Устройства применяются в составе средств и систем криптографической защиты данных для обеспечения информационной безопасности (в том числе защиты с высоким уровнем секретности) в государственных и коммерческих структурах.

Устройства КРИПТОН гарантируют защиту информации, обрабатываемой на персональном компьютере и/или передаваемой по открытым каналам связи.

Устройства КРИПТОН выполнены в виде плат расширения ISA и PCI персонального компьютера с процессором i386 и выше.

Преимущества устройств серии КРИПТОН:

1. аппаратная реализация алгоритма криптографического преобразования гарантирует целостность алгоритма;
2. шифрование производится и ключи шифрования хранятся в самой плате, а не в оперативной памяти компьютера;
3. аппаратный датчик случайных чисел;
4. загрузка ключей шифрования в устройство КРИПТОН со смарт-карт и идентификаторов Touch Memory (i-Button) производится напрямую, минуя ОЗУ и системную шину компьютера, что исключает возможность перехвата ключей;
5. на базе устройств КРИПТОН можно создавать системы защиты информации от несанкционированного доступа и разграничения доступа к компьютеру;
6. применение специализированного шифрпроцессора для выполнения криптографических преобразований разгружает центральный процессор компьютера; возможна также установка на одном компьютере нескольких устройств КРИПТОН, что еще более повысит скорость шифрования (для устройств с шиной PCI);
7. использование парафазных шин в архитектуре шифрпроцессора исключает угрозу снятия ключевой информации по возникающим в ходе криптографических преобразований колебаниям электромагнитного излучения в

цепях “земля - питание” микросхемы.

Программное обеспечение устройств КРИПТОН позволяет:

1. шифровать компьютерную информацию (файлы, группы файлов и разделы дисков), обеспечивая их конфиденциальность;
2. осуществлять электронную цифровую подпись файлов, проверяя их целостность и авторство;
3. создавать прозрачно шифруемые логические диски, максимально облегчая и упрощая работу пользователя с конфиденциальной информацией;
4. формировать криптографически защищенные виртуальные сети, шифровать IP-трафик и обеспечивать защищенный доступ к ресурсам сети мобильных и удаленных пользователей;
5. создавать системы защиты информации от несанкционированного доступа и разграничения доступа к компьютеру.

##### Программный интерфейс Crypton API

Crypton API v2.25 - пакет программ, реализующих интерфейс между УКЗД серии "Криптон" и программами Win32 в операционных средах Windows NT 4.0 и Windows 95/98.

Данный пакет состоит из следующих элементов:

-драйвер и библиотека для поддержки DOS приложений в Windows NT 4.0 (CryptVDD.dll и CryptDOS.sys),

-драйвер для поддержки DOS приложений в Windows 95/98 (cryptdos.vxd),

-библиотека функций Crypton API (CryptAPI.dll),

-приложение-пример, использующий функции Crypton API (TestAPI.exe),

-библиотека SCApi, обобщающая использование карт и устройств чтения карт разных типов.

Данный пакет необходим для работы Win32-приложений, использующих УКЗД серии "Криптон". Также данная библиотека необходима при работе приложений DOS, использующих УКЗД, в ОС Windows 95/98/NT 4.0.

Библиотека Crypton API v2.25.

Оригинальная библиотека Crypton API является необходимой интерфейсной составляющей и обеспечивает обеспечивает программный интерфейс к устройствам криптографической защиты данных (УКЗД) серии [КРИПТОН](http://www.ancud.ru/catalog/crypton.htm) для приложений Win32 и DOS-программ в режиме эмуляции DOS в операционных средах Windows 95/98/NT 4.0/2000/XP/2003, Solaris 2.x, 7, 8 (x86, Sparc).

В состав данного пакета программ входят: драйверы УКЗД для Windows 95/98/NT 4.0/2000/XP/2003, драйверы поддержки DOS-приложений в режиме эмуляции DOS, Win32-приложение, тестирующее УКЗД.



Универсальность интерфейса, предоставляемого пакетом программ Crypton API, состоит в том, что предоставляемый программам интерфейс идентичен независимо от конкретного УКЗД серии КРИПТОН (или даже его программного эмулятора) и независимо от типа ключевого носителя, подключаемого через интерфейс SCApi. Эмулятор подключается к Crypton API аналогично драйверу УКЗД серии КРИПТОН.

##### Программный эмулятор Crypton Emulator

Crypton emulator v1.4 - драйвер, эмулирующий работу устройства криптографической защиты данных серии "Криптон" в операционных средах Windows NT 4.0 и Windows 95/98. Перед установкой драйвера на компьютере должен быть установлен Crypton API версии 2.25 или выше.

##### Практическое занятие № 5. Изучение криптографических функций защиты данных устройств серии Криптон

* 1. Цель работы

Целью работы является изучение принципов работы программно-аппаратного комплекса Криптон.

##### Краткие теоретические сведения

Интерфейс прикладных программ Crypton DK - это динамически подключаемая библиотека (DLL) функций для работы со средствами криптографической защиты данных (СКЗД) КРИПТОН/Crypton на платформе Win32. Она предназначена для независимых разработчиков приложений, требующих встраивания функций шифрования с помощью шифраторов серии КРИПТОН/Crypton.

Crypton DK предоставляет функции шифрования по симметричной ключевой системе. Аппаратный или программный шифраторы не входят в комплект поставки Crypton DK.

Использование библиотеки Crypton DK дает возможность компаниям-разработчикам приложений, не обладающим лицензией ФСБ на распространение криптосредств, реализовать в своих программах криптографические функции.

Ознакомившись с описанием Crypton DK и убедившись, что реализуемые функции шифрования соответствуют необходимым, компания-разработчик может приобрести Crypton DK и шифратор - программный [Crypton](http://www.ancud.ru/catalog/emulator.htm) [Emulator](http://www.ancud.ru/catalog/emulator.htm) или аппаратный [КРИПТОН](http://www.ancud.ru/catalog/crypton.htm). С помощью Crypton DK организуется вызов криптографических функций в программе, которая получает возможность шифровать обрабатываемую ею информацию на программном или аппаратном шифраторе.

При последующем распространении программы шифратор может поставляться самой компанией- разработчиком, если у нее есть соответствующая лицензия ФСБ, либо Фирмой АНКАД напрямую пользователям программы. В последнем случае ответственность за реализацию шифросредств не лежит на компании-разработчике.

##### Crypton DK состоит из следующих элементов:

* библиотека CryptAPI, реализующая программный интерфейс CryptonAPI;
* заголовочные файлы для программ на языке С;
* примеры программ на языке С;
* man-страницы с описанием всех функций CryptonAPI;
* аппаратный ключ.

Драйверы для работы СКЗД в ОС Windows Crypton API поставляются в комплекте вместе с СКЗД

Crypton DK содержит описание Windows-аналогов всех функций BIOS КРИПТОН и примеры их использования. DOS-программы могут вызвать эти функции по прерыванию 4Сh.

Библиотека CryptonDK защищена от копирования с помощью аппаратного ключа,подключаемого к параллельному порту компьютера.Таким образом, программный продукт независимого разработчика, использующий функции библиотеки шифрования, оказывается автоматически защищенным от копирования. Аппаратные ключи защиты от копирования прозрачны для других устройств, использующих параллельный порт,например принтеров. Кроме того, ключи позволяют подключать их друг за другом, что позволяет применять на одном компьютере защищенные программные продукты разных производителей.

##### Обращение приложения Win32 к устройству шифрования:

1. Уровень приложений (не зависит от ОС)

Win32App1.exe Win32App2.exe ... Win32AppN.exe

##### Уровень, обеспечивающий интерфейс приложений с драйвером (не зависит от ОС)

CryptAPI.dll

##### Уровень ядра ОС (в зависимости от ОС используются два драйвера)

CRYPTON.SYS (Windows NT) CRYPTON.VXD (Windows 95)

##### Физический уровень

Плата шифрования

Драйвер платы шифрования виртуализует плату шифрования, т.е.каждое Win32 приложение имеет собственную виртуальную плату шифрования со своими ключами К1 и К2, однако ключ К3 и узел замены являются общими для всех приложений. Уровни 3 и 4 могут эмулироваться драйверами- эмуляторами CRYPTONL.SYS (Windows NT) и CRYPTONL.VXD (Windows 95).

##### Поддержка DOS

DOS-программы могут использовать прерывание 0x4c в режиме эмуляции DOS. Вы можете отключить драйвер поддержки DOS сессии с помощью утилиты “DOS driver setup” (CryptDOS.exe). С помощью этой же утилиты вы сможете выбрать какой драйвер (работающий с платой шифрования или эмулятор) будет использоваться в прерывании 0x4C.

Обращение DOS-приложения к устройству шифрования в Windows: При обращении к устройству запрос от MS-DOS приложения проходит несколько уровней:

##### Уровень приложений MS-DOS

V86App1.exe V86App2.exe ... V86AppN.exe

##### Уровень, обеспечивающий интерфейс программ реального режима с функциями защищенного режима

cryptdos.vxd

##### Уровень ядра ОС

crypton.vxd

##### Физический уровень

Плата шифрования

Когда стартует DOS-сессия, автоматически стартует драйвер cryptdos.vxd. Этот драйвер перехватывает прерывание 0x4С. Когда DOS-приложение генерирует программное прерывание 0x4C, драйвер передает управление драйверу платы шифрования crypton.vxd.

Описание DOS-функций BIOS КРИПТОН находится в руководстве программиста (файл program.txt), поставляемом в комплекте СКЗД "Криптон-4".

##### Порядок выполнения работы

УСТАНОВКА ПРОГРАМНОГО ОБЕСПЕЧЕНИЯ.

* Инсталляция Crypton API

Перейти в директорию CryptonAPI и запустить CRYPTONAPI.EXE.

Далее, в создавшемся каталоге CryptonAPI запустить программу установки install.exe. В параметрах установки выбрать устройство Crypton-4K/16 , оно стоит по умолчанию.

* Инсталляция Crypton emulator

Перейти в директорию Emulator и запустить программу установки install.exe.

* Настройка Ancud Software:

Скопируйте на дискету файлы gk.db3 и uz.db3 , которые находятся в дириктории Emulator. ГК (главный ключ

– файл gk.db3) и УЗ (узел замены – файл uz.db3) - ключевые элементы, используемые эмулятором для выполнения криптографических функций и должны быть загружены в эмулятор до начала выполнения каких- либо криптографических функций.

Далее, в меню Пуск -> Программы -> Ancund Software -> Cripton API-> запустить “конфигурация драйвера поддержки ДОС “ и выбрать в меню “Использовать драйвер” Драйвер эмулятор. (Рисунок 1)



##### Рисунок 1.

После этого, вставьте дискету, в меню Пуск -> Программы -> Ancund Software ->Crypton Emulator запустите “Загрузить ключи” (Рисунок 2)

##### Рисунок 2.

После этого всё готово к выполнению шифрования и дешифрования информации. Протестировать функции шифрования можно запустив в меню Пуск -> Программы -> Ancund Software -> Cripton API-> “Программа тестирования функций Crypton API “ (Рисунок 3)



##### Рисунок 3.

После запуска “Программа тестирования функций Crypton API “ в меню Драйвер -> Открыть выберите “Драйвер эмулятор” (Рисунок 4)

##### Рисунок 4.

Затем в меню “Шифрование” можно зашифровать и дешифровать короткое сообщение.

* Инсталляция Центра Генерации Ключей (ЦГК). Установите ЦГК (diKey.exe) запустив файл Key2.exe
* Инсталляция пакета dipost. Установите dipost запустив setup.exe из папки dipost

##### Содержание отчета

В отчете описать порядок выполнения работы. Привести значения сгенерированных ключей. Привести результат тестов устройства.

##### Контрольные вопросы

1. Что содержит файл gk
2. Что содержит файл uz
3. Назначение главного ключа
4. Рассказать о иерархии ключей

##### Практическое занятие № 31. Генерация ключевой информации и криптографические средства в клиентской программе электронной почты

DiPost является программным обеспечением рабочих мест пользователей Центров информационно- коммуникационных услуг ДИОНИС (в дальнейшем просто Центров) и других систем, поддерживающих протоколы SMTP и POP3. DiPost позволяет пользователю готовить и хранить почтовую корреспонденцию, проводить сеансы двухстороннего обмена информацией с Центрами по различным типам каналов связи. При подсоединении абонента к Центру обмен информацией u1074 выполняется автоматически: все данные, подготовленные абонентом к отправке, пересылаются в Центр, а данные, пришедшие в адрес абонента с момента его последнего сеанса работы с Центром, пересылаются на ПЭВМ пользователя. По окончании сеанса обмена данными пользователь может заниматься обработкой полученной информации.

В дополнение к обычным функциям почтовых систем в DiPost реализованы следующие возможности. Работа с несколькими почтовыми ящиками DiPost позволяет организовать для абонента несколько почтовых ящиков.

Такая возможность позволяет пользователю, зарегистрированному в разных Центрах, разделить почтовые потоки. Один почтовый ящик (Системный) создается автоматически, а все последующие должны быть созданы пользователем вручную. Работа по линиям связи различного типа DiPost позволяет подсоединиться к Центру по телефонным каналам (коммутируемым и выделенным) через модем, по каналам локальной сети, работающим по протоколам IPX, TCP/IP, SMTP

и POP3, по коммутируемой сети с использованием протоколов PPP или SLIP.

Автоматическая доставка почты В DiPost реализован режим автоматического обмена информацией с Центром через заданный интервал времени. Автоматическая подготовка исходящей почты В DiPost реализован режим автоматической конвертации файлов в почтовые сообщения и отправки их в Центр при инициировании обмена данными с Центром.

Различные форматы хранения почтовой корреспонденции DiPost поддерживает для хранения почтовых сообщений (писем, писем с файлами) 3 почтовых формата - Дионис, MIME и UUENCODE. Возможность хранения любого сообщения в формате Дионис предоставляет пользователю удобные средства для просмотра и редактирования текстов сообщений. Автоматическое преобразование формата входящей почты Для входящей почты в DiPost предусмотрен поиск писем в формате MIME и автоматическое конвертирование их в формат Дионис. При необходимости от автоматического преобразования формата можно отказаться.

Поддержка уведомлений DiPost позволяет оформить заказ уведомлений для исходящей почты и сформировать уведомления о прочтении входящей корреспонденции абонентом

##### Цель работы

Освоить работу с клиентской программой электронной почты. Изучить работу системы защищенного документооборота.

##### Краткие теоретические сведения

Система формирования и сдачи налоговой и бухгалтерской отчетности в электронном виде по каналам связи представляет собой совокупность программно-аппаратных средств, принадлежащих налогоплательщикам, налоговым органам и оператору системы (или его коммерческим представителям), а также совокупность документов, регламентирующих взаимоотношения участников системы. На рис. 1 представлена схема взаимодействия основных элементов системы

«Такском-Спринтер» при обмене электронными документами между налоговыми органами и налогоплательщиками.

Главной функцией системы «Такском-Спринтер» является осуществление обмена открытой и конфиденциальной информацией между налогоплательщиками и налоговыми органами, в том числе при передаче подписанных электронной цифровой подписью и зашифрованных электронных документов, содержащих данные бухгалтерской и налоговой отчетности или сведения об исполнении налоговых обязательств перед бюджетом.

Внедрение системы формирования и сдачи отчетности в электронном виде по каналам связи обеспечит для ее участников следующие возможности:

* общее сокращение затрат времени на подготовку и сдачу отчетности налогоплательщиком;
* оперативное обновление форм отчетности, передаваемых по каналам связи;
* сокращение количества ошибок при подготовке отчетности за счет применения программных средств, обеспечивающих проверку и контроль выходных документов;
* удобный доступ к информации об изменении бюджетных счетов, к инструктивным письмам и другим материалам, предоставляемым налоговыми органами налогоплательщикам;
* предоставление налогоплательщику (по его запросу) информационных выписок о состоянии его расчетов с бюджетами всех уровней, выдаваемых налоговой инспекцией.



Модем

Рабочее место налогоплательщика

Модем

Интернет

Интернет

Сервер электрон- ной почты

Программный комплекс инспекции ФНС

Программный комплекс оператора системы

Программный комплекс "Спринтер"

1. Программа ввода (выгрузки)
2. Почтовый клиент

3.Модуль контроля и визуализации

4. Программа "Референт"

Рабочее место налоговых органов

Рис.1. Схема взаимодействия основных элементов системы "Такском-Спринтер" при обмене электронными документами

При сдаче бухгалтерской и налоговой отчетности в налоговые органы по электронной почте в рамках системы «Такском-Спринтер» применяется специальная технология электронного документооборота, которая обеспечивает:

* защиту содержания электронных документов, циркулирующих в системе, от несанкционированного просмотра третьими лицами;
* однозначную идентификацию абонентов системы — отправителей электронных документов;
* защиту электронных документов от несанкционированных изменений;
* корректное разрешение возникающих спорных ситуаций.

Первые три проблемы решаются с использованием средств криптографической защиты информации, в состав которых входят средства электронной цифровой подписи и шифрования, а последняя — путем установления специального порядка (регламента) обмена электронными документами между налогоплательщиками и налоговыми органами.

Регламент позволяет зафиксировать в качестве даты отправки отчетности тот момент времени, когда сообщение с файлом отчетности получено на сервере электронной почты специализированного оператора связи системы для отправки в адрес инспекции. Средства отработки регламента доставляют налогоплательщику электронные документы, подтверждающие:

* что в качестве файла отчетности в налоговой инспекции будет обрабатываться именно тот файл, который отправлен налогоплательщиком по электронной почте;
* соответствие файла отчетности «Формату представления бухгалтерской и налоговой отчетности в электронном виде»;
* дату отправки отчетности.

Процедура регламента позволяет однозначно установить, кто из участников переписки несет ответственность за возникновение каждой из возможных нештатных и спорных ситуаций.

##### Порядок выполнения работы

1. Запустите программу dipost. (Рисунок 7)
2. В меню “Ящик” выберите “Параметры почтового ящика “ .В поле “Адрес электронной почты”введите адрес, состоящий из инициалов абонента, в поле “имя ключа” введите ID абонента (имя файла в папке PRIVATE на дискете), установите галочки напротив полей “ Подписывать письма” и “Шифровать письма” Например для абонента DSU (Рисунок8)

В меню “Ящик” выберите “Установки криптосистемы”, установите параметры соответствующие данному абоненту. (Рисунок 9).

Примечание, возможно необходимо будет изменить имя файла в дискете в разделе PRIVATE.



##### Рисунок 7

Рисунок 8



Рисунок 9

1. В меню “Почта” выбираем “Адресная книга” и заносим в нее адреса абонентов .Например для РАС (Рисунок 10):

##### Рисунок 10

1. Теперь мы можем отсылать зашифрованные сообщения (Рисунок 11) .



##### Рисунок 11.

Если у Вас нет выхода в Интернет , то тогда можно имитировать работу почтового агента следующим образом:

Создаём в паке абонента DSU “Исходящие” сообщение, прикрепляем какой-нибудь текстовый файл, шифруем его ( значёк замочек) .Закрываем Dipost. Затем копируем файлы из папки “Dipost\BOXES\DSU\Исходящие” в папку “Dipost\BOXES\RAS\Входящие”. Далее вставляем дискету с ключами RAS .Запускаем Dipost, в почтовом ящике RAS открываем папку “Входящие” и расшифровываем файл. Сеанс связи окончен.

Данную работу выполняют по 2 студента. Форма отчётности - подробное описания проделанной работы по каждому шагу в электронном документе (Microsoft Word).

##### Содержание отчета

Форма отчётности - подробное описания проделанной работы по каждому шагу в электронном документе (Microsoft Word). Привести структуру ключевой дискеты. Содержание файла открытых и закрытых ключей.

##### Контрольные вопросы

1. Рассказать о структуре ключевой дискеты
2. Как получают закрытый ключ?
3. Как получают открытый ключ?
4. Что такое регламент защищенного документооборота?
5. Каким базовым угрозам противостоит система защищенного документооборота

##### Содержание

Введение

 Программно-аппаратные средства криптографической защиты информации серии Криптон

* 1. Концептуальный подход к защите информации в компьютерных системах
	2. Устройства криптографической защиты данных серии криптон
	3. Программный интерфейс Crypton API
	4. Программный эмулятор Crypton Emulator

 Практическое занятие № 1. Изучение криптографических функций защиты данных устройств серии Криптон

1. Цель работы
2. Краткие теоретические сведения
3. Порядок выполнения работы
4. Содержание отчета
5. Контрольные вопросы

 Практическое занятие № 2.

1. Цель работы
2. Краткие теоретические сведения
3. Порядок выполнения работы
4. Содержание отчета
5. Контрольные вопросы

сПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

**Основная**:

1. Баранова, Е. К. Основы информационней безопасности : учебник / Е. К. Баранова, А. В. Бабаш. - Москва : РИОР : ИНФРА-М, 2021. — 202 с. — (Среднее профессиональное образование). - ISBN 978-5-369-01806-4. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1209579>

**Дополнительная:**

1. Сычев, Ю. Н. Защита информации и информационная безопасность : учебное пособие / Ю.Н. Сычев. — Москва : ИНФРА-М, 2021. — 201 с. — (Среднее профессиональное образование). - ISBN 978-5-16-016583-7. - Текст : электронный. - URL: https://znanium.com/catalog/product/1191479.