ГОУВПО
Донецкий национальный технический университет

Методические указания и задания к курсовому проекту
по дисциплине «Безопасность программ и данных»
для студентов направления подготовки «Программная инженерия»

На тему: «Разработка почтового клиента с использованием криптографических средств защиты электронной почты»

Донецк 2016

ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Методические указания и задания к курсовому проекту по дисциплине «Безопасность программ и данных» для студентов направления подготовки «Программная инженерия» На тему: «Разработка почтового клиента с использованием криптографических средств защиты электронной почты»

Рассмотрено на заседании кафедры
ПИ
Протокол № 1 от 30.08.16
Утверждено на заседании
учебно-издательского Совета ДонНТУ
протокол № от

Донецк 2016

УДК 681.3

Методические указания и задания к курсовому проекту по дисциплине «Безопасность программ и данных» для студентов направления подготовки «Программная инженерия»/ Сост.: Чернышова А.В. - Донецк, ДонНТУ, 2016 - 55 стр.

Приведены методические указания и задания к выполнению курсового проекта по курсу «Безопасность программ и данных» для студентов направления подготовки «Программная инженерия». Излагаются вопросы, связанные с криптографическими средствами защиты почтовых клиентов.

Методические указания предназначены для усвоения теоретических основ и формирования практических навыков по курсу «Безопасность программ и данных».

Составители: Ст.преп. каф. ПИ Чернышова А.В.

Рецензент: доцент кафедры КМД Губенко Н.Е.

Методические указания к выполнению курсового проекта

Прежде, чем рассматривать протоколы передачи почты, рассмотрим компоненты почтовой системы. На рисунке 1 представлены реальные компоненты почтовой системы Интернет.

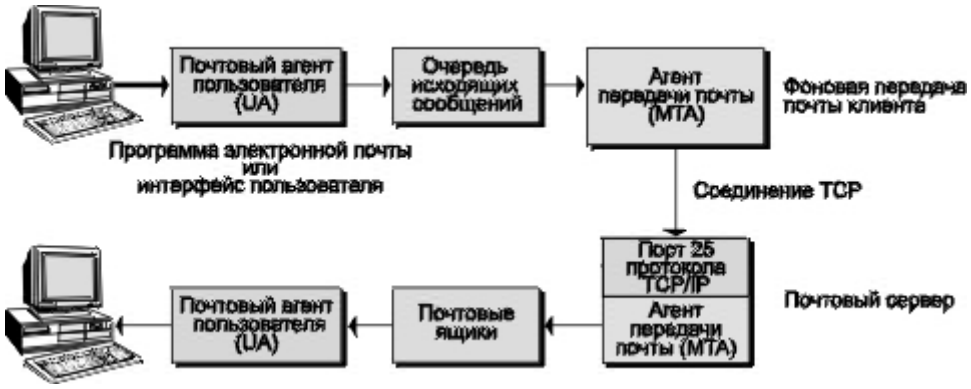

Рисунок 1 - Компоненты почтовой системы

Термин «агент» довольно часто встречается в документации Интернет. «Агент»— это программа специального назначения, выполняющая действия для пользователя или другой программы. В большинстве случаев почтовая программа называется агентом пользователя (UA). Точно так же агент передачи почты (MTA) представляет собой клиент или сервер, выполняющий задачи по доставке или получению почты на сетевом компьютере.

Вы, как пользователь, взаимодействуете с агентом пользователя. Он, в свою очередь, взаимодействует с файлом-контейнером или агентом передачи сообщений за вас. В то же время, MTA ведет себя как представитель своего компьютера в сети. Агент пользователя защищает вас от необходимости общаться с различными почтовыми хостами, а MTA защищает компьютер от необходимости общаться с различными агентами пользователя или несколькими агентами передачи почты одновременно.

Пользовательский агент отделен от агента передачи почты. Конечно, их можно объединить в одной программе, но все равно это будут отдельные модули. Будучи взаимосвязаны, оба агента выполняют совершенно различные функции.

Система электронной почты представлена агентами передачи почты, MTA. До того как обсудить задачи пользовательского агента, необходимо узнать

немного больше о том, что же такое MTA. MTA умеют устанавливать TCP-соединение для связи с другими MTA. Протоколом этого соединения, как правило, является простой протокол передачи почты (SMTP).

Простой протокол передачи почты (SMTP)

Агент передачи почты— основной компонент системы передачи почты Интернет. Как уже говорилось, MTA как бы представляет данный сетевой компьютер для сетевой системы электронной почты. Пользователи редко имеют дело с MTA, поскольку он не вполне «дружелюбен», однако без него не обходится ни одна почтовая система. После того как UA пошлет сообщение в выходную очередь, за дело принимается MTA. Он извлекает сообщение и посылает его другому MTA. Этот процесс продолжается до тех пор, пока сообщение не достигнет компьютера-получателя. Для передачи сообщений по TCP-соединению большинство MTA пользуются протоколом SMTP. Сообщения форматированы по правилам виртуального сетевого терминала (NVT), то есть в NVT ASCII. NVT подобен виртуальному сетевому протоколу и нужен затем, чтобы скрыть различия в восприятии разными компьютерами разных символов, например переводов каретки, переводов строки, маркеров конца строки, очистки экрана и т.д. Символ в NVT состоит из семи битов набора ASCII и является буквой, цифрой или знаком пунктуации. Семибитный набор ASCII часто называется NVT ASCII.

Команды SMTP

Простой протокол передачи почты обеспечивает двухсторонний обмен сообщениями между локальным клиентом и удаленным сервером MTA. MTA-клиент шлет команды MTA-серверу, а он, в свою очередь, отвечает клиенту. Другими словами, протокол SMTP требует получать ответы от приемника команд SMTP. Обмен командами и ответами на них называется почтовой транзакцией (mail transaction). Данные передаются в формате NVT ASCII. Кроме того, команды тоже передаются в формате NVT ASCII. Команды

передаются в форме ключевых слов, а не специальных символов, и указывают на необходимость совершить ту или иную операцию. В табл. 1 приведен список ключевых слов (команд), определенный в спецификации SMTP— RFC 821.

Таблица 1 – Команды простого протокола передачи почты (SMTP).

Таблица 15.1. Команды простого протокола передачи почты (SMTP)

Команда	Обязательна	Описание
HELO	X	Идентифицирует модуль-передатчик для модуля-приемника (hello).
MAIL	X	Начинает почтовую транзакцию, которая завершается передачей данных в один или несколько почтовых ящиков (mail).
RCPT	X	Идентифицирует получателя почтового сообщения (recipient).
DATA		Строки, следующие за этой командой, рассматриваются получателем как данные почтового сообщения. В случае SMTP, почтовое сообщение заканчивается комбинацией символов: CRLF-точка-CRLF.
RSET		Прерывает текущую почтовую транзакцию (reset).
NOOP		Требует от получателя не предпринимать никаких действий, а только выдать ответ OK. Используется главным образом для тестирования. (No operation.)
QUIT		Требует выдать ответ OK и закрыть текущее соединение.
VRFY		Требует от приемника подтвердить, что ее аргумент является действительным именем пользователя. (См. примечание.)
SEND		Начинает почтовую транзакцию, доставляющую данные на один или несколько терминалов (а не в почтовый ящик).
SOML		Начинает транзакцию MAIL или SEND, доставляющую данные на один или несколько терминалов или в почтовые ящики.
SAML		Начинает транзакцию MAIL и SEND, доставляющие данные на один или несколько терминалов и в почтовые ящики.
EXPN		Команда SMTP-приемнику подтвердить, действительно ли аргумент является адресом почтовой рассылки и если да, вернуть адрес получателя сообщения (expand).
HELP		Команда SMTP-приемнику вернуть сообщение-справку о его командах.
TURN		Команда SMTP-приемнику либо сказать OK и поменяться ролями, то есть стать STMP-передатчиком, либо послать сообщение-отказ и остаться в роли SMTP-приемника.

В RFC 821 сказано, что команда VRFY не является обязательной для минимального набора команд SMTP. Однако в RFC 1123 «Требования для сетевых компьютеров Интернет— приложения и обеспечение работы» (Requirements for Internet Hosts— Application and Support, Braden, 1989), команда

VRFY фигурирует в списке обязательных для Интернет команд реализации SMTP.

В соответствии со спецификацией команды, помеченные крестиком (X) в табл. 1, обязаны присутствовать в любой реализации SMTP. Остальные команды SMTP могут быть реализованы дополнительно. Каждая SMTP-команда должна заканчиваться либо пробелом (если у нее есть аргумент), либо комбинацией CRLF. В описании команд употреблялось слово «данные», а не «сообщение». Этим подчеркивалось, что, кроме текста, SMTP позволяет передавать и двоичную информацию, например графические или звуковые файлы. Другими словами, SMTP способен передавать данные любого содержания, а не только текстовые сообщения. Это значит, что, рассматривая вопросы, касающиеся SMTP, не забывайте, что термин «сообщение» обозначает не только текстовые данные.

Во многих операционных системах для системного администратора предусмотрена возможность послать сообщение многим пользователям одновременно. Например, перед остановкой системы администратор выполняет команду, которая автоматически рассылает предупреждающее сообщение на каждый подключенный терминал. С другой стороны, пользователь может отключить прием таких сообщений. Тот или иной режим приема сообщений устанавливается при входе пользователя в систему по умолчанию и может измениться позже.

Во многих компьютерах механизм рассылки таких сообщений подобен системе электронной почты. В SMTP существуют команды, позволяющие доставлять сообщения двумя путями— в почтовый ящик или на терминал. В терминах SMTP команда send означает «послать на терминал», а команда mail— «послать в почтовый ящик». Команда SEND является дополнительной, тогда как MAIL— обязательна в любой реализации.

Коды ответов SMTP

В спецификации SMTP требуется, чтобы сервер отвечал на каждую

команду SMTP-клиента. MTA-сервер отвечает трехзначной комбинацией цифр, называемой кодом ответа. Вместе с кодом ответа, как правило, передается одна или несколько строк текстовой информации.

Несколько строк текста, как правило, сопровождают только команды EXPN и HELP. В спецификации SMTP, однако, ответ на любую команду может состоять из нескольких строк текста.

Каждая цифра в коде ответа имеет определенный смысл. Первая цифра означает, было ли выполнение команды успешно (2), неуспешно (5) или еще не закончилось (3). Как указано в приложении E документа RFC 821, простой SMTP-клиент может анализировать только первую цифру в ответе сервера, и на основании ее продолжать свои действия. Вторая и третья цифры кода ответа разъясняют значение первой. Если вы разрабатываете SMTP-приложение, обязательно изучите конструкцию всех кодов SMTP-ответа.

На рисунке 2 представлены коды ответа SMTP и их значение.

Код	Значение
211	Ответ о состоянии системы или помощь
214	Сообщение-подсказка (помощь)
220	<имя_домена> служба готова к работе
221	<имя_домена> служба закрывает канал связи
250	Запрошенное действие почтовой транзакции успешно завершилось
251	Данный адресат не является местным; сообщение будет передано по маршруту <forward-path>
354	Начинай передачу сообщения. Сообщение заканчивается комбинацией CRLF-точка-CRLF
421	<имя_домена> служба недоступна; соединение закрывается
450	Запрошенная команда почтовой транзакции не выполнена, так как почтовый ящик недоступен
451	Запрошенная команда не выполнена; произошла локальная ошибка при обработке сообщения
452	Запрошенная команда не выполнена; системе не хватило ресурсов
500	Синтаксическая ошибка в тексте команды; команда не опознана
501	Синтаксическая ошибка в аргументах или параметрах команды
502	Данная команда не реализована
503	Неверная последовательность команд
504	У данной команды не может быть аргументов
550	Запрошенная команда не выполнена, так как почтовый ящик недоступен
551	Данный адресат не является местным; попробуйте передать сообщение по маршруту <forward-path>
552	Запрошенная команда почтовой транзакции прервана; дисковое пространство, доступное системе, переполнилось
553	Запрошенная команда не выполнена; указано недопустимое имя почтового ящика
554	Транзакция не выполнена

Рисунок 2 - коды ответа SMTP и их значение

Ограничения по размерам

В стандарте SMTP сказано, что реализации SMTP не должны ограничивать максимальную длину обрабатываемых объектов (возможно, для будущих расширений стандарта). Однако в настоящий момент SMTP ограничивает допустимые размеры следующими величинами, приведенными в табл. 2.

Таблица 2 – Ограничение на размеры объектов SMTP

Таблица 15.3. Ограничения на размеры объектов SMTP

Объект SMTP	Ограничение
User	Максимальная длина имени пользователя: 64 символа
Domain	Максимальная длина имени домена: 64 символа
Path	Максимальная длина обратного маршрута или маршрута доставки, включая знаки пунктуации и символы-ограничители: 256 знаков
Command line	Максимальная длина командной строки, включая ключевое слово и символы CRLF: 512 знаков
Reply line	Максимальная длина строки ответа, включая код ответа и символы CRLF: 512 знаков
Text line	Максимальная длина текстовой строки, включая символы CRLF: 1000 знаков

В соответствии со спецификацией (RFC 821), если клиент MTA превысил ограничения на размер передаваемой информации, сервер MTA отвечает одним из следующих кодов:

500 Line too long.(Слишком длинная строка)

501 Path too long.(Слишком длинный путь)

552 Too many recipients.(Слишком много получателей)

552 Too much mail data.(Слишком много данных в сообщении)

Составные части сообщения электронной почты.

Сообщение электронной почты состоит из следующих частей: упаковки (envelope), для распознавания агентами передачи почты (MTA), заголовков (headers), используемых агентами пользователя (UA), и тела (body) сообщения— собственно информации, предназначенной адресату. Данные упаковки задаются полями «TO:» и «FROM:». (при рассмотрении SMTP-команды SEND и RCPT.) Другими словами, как указано в RFC 821, информация упаковки требуется для передачи сообщения от одного сетевого компьютера другому.

В RFC 822, «Стандарт оформления текстовых сообщений Интернет» (Standard for the Format of ARPA Internet Text Messages, Crocker, 1982), определены форматы двух остальных частей: заголовка и тела сообщения. (В профессиональной среде текстовые сообщения этого формата иногда

называются просто «822». Например, можно встретить заявление типа: «В этом протоколе используется текст формата 822».) Для иллюстрации сложного заголовка сообщения в RFC 822 приводится следующий пример (рисунок 3):

Date : 27 Aug 76 0932 PDT
From : Ken Davis <KDavis@This-Host.This-net>
Subject: Re: The Syntax in the RFC
Sender : KSecy@Other-Host
Reply-To: Sam.Irving@Reg.Organization
To : George Jones <Group@Some-Reg.An-Org>,
Al.Neuman@MAD.Publisher
cc : Important folk: Tom Softwood <Balsa@Tree.Root>,
"Sam Irving"@Other-Host;,
Standard Distribution: /main/davis/people/standard@Other-Host,
"<Jones>standard.dist.3"@Tops-20-Host>;
Comment : Sam is away on business.
He asked me to handle his mail for him.
He'll be able to provide a more accurate explanation when he
returns next week.
In-Reply-To: <some.string@DBM.Group>, George's message
X-Special-action: This is a sample of user-defined field-names.
There could also be a field-name "Special-action" but, its name
might later be preempted
Message-ID: 4231.629.XYzi-What@Other-Host

Рисунок 3 – Пример сложного заголовка письма

Как уже отмечалось, тело сообщения содержит собственно информацию, которую необходимо доставить адресату. Как правило, оно состоит из текста формата NVT ASCII. Заголовок сообщения отделяется от тела пустой строкой.

Тело сообщения вводится пользователем в ходе почтовой транзакции при помощи агента пользователя (почтовой программы). Далее, агент пользователя добавляет необходимые поля заголовка, содержимое которых задается пользователем, и передает сообщение следующему агенту MTA, который осуществляет либо промежуточную, либо окончательную доставку.

Расширения SMTP.

Самым важным документом, посвященным расширениям SMTP, является RFC 1425, «Расширения службы SMTP» (SMTP Service Extensions, Klensin et al,

1993). Вдобавок к расширениям SMTP, в RFC 1425 описываются способы проектирования новых расширений, так сказать, на будущее. Реализация SMTP, работающая в соответствии с расширениями RFC 1425, называется «расширенный SMTP» или просто ESMTP (Extended SMTP). Расширенный SMTP работает почти так же, как и обычный SMTP, однако команда-приветствие у него другая: EHLO (Extended hello).

Чтобы выяснить, поддерживает ли MTA-сервер спецификацию ESMTP, MTA-клиент посылает команду EHLO. Если сервер поддерживает ESMTP, он отвечает кодом 250. Если нет, следует сообщение о синтаксической ошибке. В ответ на сообщение об ошибке, клиент может выдать обычную команду HELO и далее выполнять стандартные операции SMTP. Если сервер умеет обслуживать ESMTP, в ответ на приветствие, как правило, он выдает многострочный ответ (в известном вам формате). Каждая срока ответа содержит дополнительную команду ESMTP, с которой сервер знает, как работать. Вот пример реакции ESMTP-сервера в ответ на команду EHLO:

250-mail.server.com

250-EXPN

250-HELP

250 TURN

Вторая, третья и четвертая строки ответа содержат названия дополнительных команд сервера. Этот сервер обеспечивает обработку перечисленных дополнительных команд. На самом деле в качестве расширений RFC 1425 перечислены только дополнительные SMTP-команды. Другими словами, первыми шестью расширениями SMTP в RFC 1425 являются команды: EXPN, HELP, TURN, SEND, SOML и SAML. Существует также расширение SIZE, описанное в RFC 1427, «Расширения службы SMTP, касающиеся размеров сообщений» (SMTP Service Extension for Message Size Declaration, Klensin et al, 1993), позволяющее SMTP-клиенту и серверу сообщать друг другу размеры передаваемого сообщения. Если в ответе на команду EHLO присутствует ключевое слово SIZE, значит, данное расширение обрабатывается.

Как известно, существует предел размеров передаваемого сообщения для сервера. Если клиент MTA попытается передать сообщение, превышающее этот предел, почтовая транзакция не состоится (закончится с ошибкой). Максимальная длина сообщения ограничивается по нескольким причинам. Основная состоит в том, что размеры жестких дисков сервера всегда ограничены, и слишком длинное сообщение может попросту не поместиться на них. С другой стороны, свободное пространство на дисках сервера может со временем увеличиться, и это сообщение будет принято при следующей попытке.

MTA-сервер ESMTP анализирует аргумент SIZE и решает, принимать ему сообщение такой длины или сообщить об ошибке. Все это происходит еще до начала передачи (однако в RFC 1427 описано несколько случаев, когда ошибка происходит уже во время передачи). Остальные несколько расширений SMTP, в общем, подчиняются тем же правилам, что и рассмотренная только что команда SIZE. То есть команда-расширение выдается в ответе сервера по запросу клиента EHLO. Расширения можно использовать в ваших программах в соответствии со спецификацией. В некоторых случаях расширение является просто дополнительной возможностью. В других случаях расширение— дополнительный аргумент к существующей команде (как в случае только что рассмотренной команды SIZE).

Местные расширения.

По определению местным расширением является любое поле, команда или название опции, начинающееся с буквы X.

```
X-Special-action: This is a sample of user-defined field-names.
There could also be a field-name "Special-action", but its name
might later be preempted
```

Имя этого поля начинается с X, поэтому оно местное, то есть задано

пользователем. Разумеется, пользователь может придумать любое название поля, какое ему нравится. Однако при этом существует риск, что в дальнейших стандартах Интернет будет определено то же самое имя, но с другим значением. При этом пользовательское и стандартное приложения вступят в конфликт. Поэтому при разработке собственных расширений почтовой системы необходимо, чтобы имена всех новых объектов начинались с буквы X. Например, пользовательский вариант декодирования тела сообщения должен называться не DECODE, как можно было бы предположить, а XDECODE. SMTP-сервер организации при этом должен включить местное расширение XDECODE в список, выдаваемый по команде EHLO (с кодом ответа 250):

250 XDECODE

MIME

В документе RFC 1521 под названием «Многоцелевые расширения почтовой системы Интернет» (MIME, Multipurpose Internet Mail Extensions, Borenstein and Bellcore, 1993) описано наиболее впечатляющее расширение для существующих почтовых систем. Система MIME не предполагает вмешательства в деятельность агентов передачи почты. Два агента пользователя, понимающие MIME, могут общаться друг с другом при помощи обыкновенных MTA. В MIME к сообщению просто добавляются несколько полей заголовка (в соответствии с RFC 822):

MIME-Version (версия MIME)

Content-Type (тип содержимого)

Content-Transfer-Encoding (тип кодировки содержимого)

Content-ID (идентификатор содержимого)

Content-Description (описание содержимого)

Номера версий MIME меняются по мере его развития. Поле MIME-Version задает номер версии расширения MIME, которое данный агент пользователя умеет обрабатывать. Номер версии в заголовке предохраняет

агента от неправильной интерпретации сообщения, в случае, если версии MIME сообщения и агента не совпадают. Вот образец полей заголовка MIME-Version и Content-Type:

Mime-Version: 1.0Content-Type: TEXT/PLAIN;

charset=US-ASCII

Как видим, сообщение создано MIME версии 1.0. Тип содержимого— TEXT, подтип— PLAIN, кодовая таблица (набор символов) US-ASCII. В табл. 3 приведены существующие в данный момент типы и подтипы MIME.

Таблица 3 – Существующие типы и подтипы MIME

Таблица 15.4. Существующие типы и подтипы MIME

Тип	Подтип	Описание
Text	Plain	Неформатированный текст.
	Richtext	Текст с элементами форматирования и выделениями, например с курсивом, подчеркиваниями, жирными буквами и т. д.
	Enriched	Усовершенствованный и упрощенный вариант подтипа richtext.
Multipart	Mixed	Тело сообщения состоит из нескольких частей; обрабатывать последовательно.
	Parallel	Тело сообщения состоит из нескольких частей; обрабатывать параллельно.
	Digest	Дайджест электронной почты.
	Alternative	Тело сообщения состоит из нескольких частей; все части семантически идентичны.
Message	RFC822	В теле содержится почтовое сообщение стандарта RFC 822.
	Partial	Фрагмент почтового сообщения.
	External-Body	Указатель на действительное почтовое сообщение (не включенное в тело данного сообщения).
Application	Octet-Stream	Произвольные двоичные данные.
	Postscript	Программа на языке Postscript.
Image	JPEG	Формат ISO 10918.
	GIF	Графический формат фирмы Compuserve.
Audio	Basic	Звук в 8-битном ISDN-формате mu-law.
Video	MPEG	Формат ISO 11172.

Поля заголовка Content-ID и Content-Description могут отсутствовать. Первое служит для идентификации MIME-содержимого электронного письма, а второе может содержать дополнительное описание. Например, если MIME-содержимым является графический образ, в поле Content-Description можно поместить описание этого образа.

В табл. 4 перечислены возможные значения поля Content-Tranfer-Encoding, доступные в настоящее время.

Таблица 4 - возможные значения поля Content-Tranfer-Encoding

Таблица 15.5. Допустимые значения поля Content-Transfer-Encoding

Content-Transfer-Encoding	Описание
7bit	Формат NVT-ASCII – стандартный формат почтовых сообщений.
Quoted-printable	Полезен в случае, если текст содержит небольшое количество восьмибитных символов.
Base64	Формат, в котором три байта данных упакованы в четыре шестибитных значения.
8bit	Содержит текст, в котором не все символы принадлежат стандартному набору ASCII (то есть, в некоторых установлен восьмой бит).
Binary	Восьмибитные данные, как правило, без символов окончания строки.

По умолчанию формат почтовых сообщений удовлетворяет кодовому набору NVT-ASCII. 8-битные агенты MTA сейчас практически отсутствуют, но как только они получат широкое распространение, вероятно, передача бинарной и текстовой информации в 8-битной кодировке возрастет. В настоящий момент для передачи 8-битной информации по 7-битным каналам Интернет лучше всего использовать кодировки quoted-printable или base64.

Способы кодирования MIME

Для кодирования небольшого количества 8-битных данных в 7-битный формат NVT ASCII лучше всего подходит схема quoted printable. 8-битный символ в этой схеме представляется в виде последовательности из трех символов. Последовательность всегда начинается со знака «равно» (=). Сразу за знаком «равно» следует двузначное шестнадцатиричное число, представляющее код ASCII кодируемого символа. Рассмотрим закодированную quoted printable последовательность JAMSA PRESS. Хоть она и не содержит 8-битных символов, зато позволяет хорошо проиллюстрировать сам принцип кодирования. Закодированное сочетание JAMSA PRESS выглядит так:

=4A=41=4D=53=41=20=50=52=45=53=53

Другими словами, буква J имеет шестнадцатиричный код ASCII 0x4A, буква A— 0x41 и т.д. Обратите внимание на то, что схема quoted printable

передает ASCII код для каждого символа последовательности. То есть для знака A (ASCII 0x4A) передается код знака «равно» (ASCII 0x3D), код цифры 4 (ASCII 0x34) и код знака A (0x41). Данную схему довольно удобно использовать, но, к сожалению, она утраивает общее количество информации в сообщении. Таким образом, область применения quoted printable— сообщение с небольшим количеством символов, в которых установлен старший (восьмой) бит. Основная часть сообщения должна состоять все-таки из обычных 7-битных символов.

Схема quoted printable годится только для небольшого количества символов с установленным восьмым битом.

В отличие от quoted printable, кодирование Base-64 увеличивает размер сообщения всего лишь на одну треть. Каждая последовательность из трех байтов (24бита) превращается в четыре шестибитовых значения (опять-таки, 24 бита). Шестибитные символы соответствуют формату NVT ASCII и приведены в таблице 5.

Таблица 5 - Таблица кодировки Base-64

6 бит	ASCII	6 бит	ASCII	6 бит	ASCII	6 бит	ASCII
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Способ кодирования Base-64 идентичен способу, описанному в RFC 1421 (Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, Linn, 1993), где рассматриваются приложения

шифрования электронной почты (Privacy Enhanced Mail applications, PEM).

Если количество байтов (символов) в сообщении не кратно трем, используется дополняющий символ «равно». На рис. 4 приведены три примера кодирования методом Base-64.

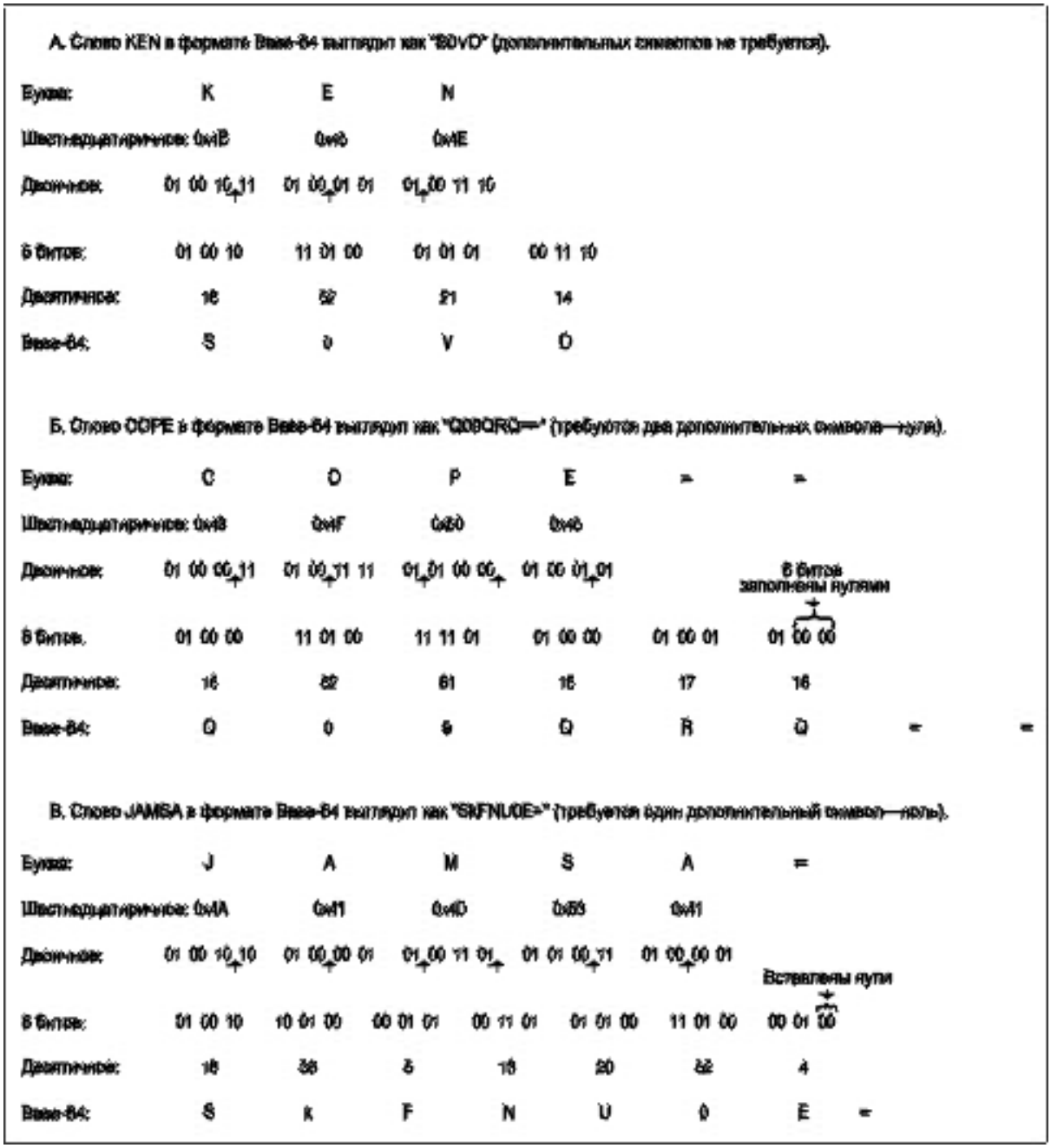

Рисунок 4 - Образцы кодирования методом Base-64 с дополнением

Протокол передачи почты POP3

В некоторых небольших узлах Интернет бывает непрактично поддерживать систему передачи сообщений (MTS - Message Transport System).

Рабочая станция может не иметь достаточных ресурсов для обеспечения непрерывной работы SMTP-сервера [RFC-821]. Для “домашних ЭВМ” слишком дорого поддерживать связь с Интернет круглые сутки.

Но доступ к электронной почте необходим как для таких малых узлов, так и индивидуальных ЭВМ. Для решения этой проблемы разработан протокол POP3 (Post Office Protocol - Version 3, STD- 53. M. Rose, RFC-1939). Этот протокол обеспечивает доступ узла к базовому почтовому серверу.

POP3 не ставит целью предоставление широкого списка манипуляций с почтой. Почтовые сообщения принимаются почтовым сервером и сохраняются там, пока на рабочей станции клиента не будет запущено приложение POP3. Это приложение устанавливает соединение с сервером и забирает сообщения оттуда. Почтовые сообщения на сервере удаляются.

Более продвинутый и сложный протокол IMAP4 обсуждается в RFC-2060 (порт 143). Об аутентификации в POP3 можно прочесть в документе RFC-1734.

В дальнейшем ЭВМ-клиентом будет называться машина, пользующаяся услугами POP3, а ЭВМ-сервером - сторона, предлагающая услуги POP3.

Когда пользователь ЭВМ-клиента хочет послать сообщение, он устанавливает SMTP связь с почтовым сервером непосредственно и посылает все, что нужно через него. При этом ЭВМ POP3-сервер не обязательно является почтовым сервером.

В исходный момент ЭВМ POP3-сервер прослушивает TCP-порт 110. Если ЭВМ-клиент хочет воспользоваться услугами POP3-сервера, то устанавливает с ним TCP связь. По установлении связи POP3-сервер посылает клиенту уведомление (например, +OK POP3 server ready) и сессия переходит в фазу авторизации (см. также RFC-1734, -1957). После этого может производиться обмен командами и откликами.

Команды POP3 состоят из ключевых слов (3-4 символа), за которыми могут следовать аргументы. Каждая команда завершается парой символов CRLF. Как ключевые слова, так и аргументы могут содержать только

печатаемые ASCII-символы. В качестве разделителя используются символы пробела. Каждый аргумент может содержать до 40 символов.

Сигнал отклика в POP3 содержит индикатор состояния и ключевое слово, за которым может следовать дополнительная информация. Отклик также завершается кодовой последовательностью CRLF. Длина отклика не превышает 512 символов, включая CRLF. Существует два индикатора состояния: положительный - "+OK" и отрицательный "-ERR" (все символы прописные).

Отклики на некоторые команды могут содержать несколько строк. В этом случае последняя строка содержит код завершения 046 ("."), за которым следует CRLF.

На практике многострочные отклики для исключения имитации завершаются последовательностью CRLF.CRLF.

процессе авторизации клиент должен представить себя серверу, передав имя и пароль (возможен вариант посылки команды APOP). Если авторизация успешно завершена, сессия переходит в состояние транзакции (TRANSACTION). При получении от клиента команды QUIT сессия переходит в состояние UPDATE, при этом все ресурсы освобождаются и TCP связь разрывается.

На синтаксически неузнанные и неверные команды, сервер реагирует, посылая отрицательный индикатор состояния.

POP3 сервер может быть снабжен таймером пассивного состояния (10 мин.), который осуществляет автоматическое прерывание сессии. Приход любой команды со стороны клиента сбрасывает этот таймер в нуль.

Сервер нумерует все передаваемые сообщения из своего почтового ящика и определяет их длину. Положительный отклик начинается с +OK, за ним следует пробел, номер сообщения, еще один пробел и длина сообщения в октетах. Завершается отклик последовательностью CRLF. Переданные сообщения удаляются из почтового ящика сервера. Все сообщения, передаваемые во время сессии POP3 должны следовать рекомендациям формата Интернет сообщений [RFC822].

В состоянии транзакции клиент может посылать серверу последовательность POP3 команд, на каждую из которых сервер должен послать отклик. Далее следует краткое описание команд, используемых в состоянии транзакция.

LIST [сообщение]

Аргументы: номер сообщения (опционно), который не может относиться к сообщению, помеченному как удаленное. Команда может быть выдана только в режиме TRANSACTION. При наличии аргумента сервер выдает положительный отклик, содержащий информационную строку сообщения. Такая строка называется скэн-листингом сообщения (scan listing). scan listing состоит из номера сообщения, за которым следует пробел и число октетов в сообщении. Сообщения, помеченные как удаленные, не пересылаются. Примером отрицательного отклика может служить: -ERR no such message.

Примеры использования команды LIST:

Клиент выдает команду: LIST

Сервер откликается: +OK 2 messages (320 octets)

Сервер: 1 120

Сервер: 2 200

Сервер: .

...

Клиент: LIST 2

Сервер: +OK 2 200

...

К: LIST 3

С: -ERR no such message, only 2 messages in maildrop

Здесь и далее символом К обозначается клиент, а символом С - сервер.

STAT - аргументов не использует, возможный отклик +OK nn mm, где nn - номер сообщения, а mm - его длина в байтах. Пример использования:

К: STAT

С: +OK 2 320

QUIT - аргументов не использует, возможный отклик +OK.

Сервер POP3 удаляет все сообщения, помеченные как удаленные из почтового ящика, посылает соответствующий отклик и разрывает TCP связь. Пример:

К: QUIT

С: +OK dewey POP3 server signing off.

RETR msg (msg - номер сообщения)

Если POP3-сервер выдал положительный отклик, то за начальным +OK следует сообщение с номером, указанным в аргументе. Отрицательный отклик имеет вид -ERR no such message.

Пример использования команды:

К: RETR 1

С: +OK 120 octets

С:

С: .

DELE msg (msg - номер сообщения)

Сервер POP3 помечает сообщение как удаленное. Любая ссылка на это сообщение в будущем вызовет ошибку. При этом само сообщение не удаляется пока сессия не войдет в режим UPDATE.

Пример использования команды:

К: DELE 1

С: +OK message 1 deleted

...

К: DELE 2

С: -ERR message 2 already deleted

NOOP (не использует каких-либо аргументов). При реализации этой команды сервер не делает ничего, лишь посылает положительный отклик.

RSET (не использует каких-либо аргументов)

Если какие-либо сообщения помечены как удаленные, сервер POP3 удаляет эту пометку и возвращает положительный отклик. Например:

К: RSET

С: +OK maildrop has 2 messages (320 octets)

Если сессия завершается не по команде клиента, то перехода в состояние UPDATE не производится, а сообщения не удаляются из почтового ящика. Далее следует описание команд, используемых в состоянии UPDATE.

Ряд команд не входят в перечень обязательных (являются опционными).

TOP msg n, где msg - номер сообщения, а n - число строк (используется только в режиме TRANSACTION).

При положительном отклике на команду TOP сервер посылает заголовки сообщений и вслед за ними n строк их текста. Если n больше числа строк в сообщении, посылается все сообщение.

UIDL [msg], где msg - номер сообщения является опционным (Unique-ID Listing).

Если сервер выдаст положительный отклик, будет выдана строка, содержащая информацию о данном сообщении. Эта строка называется уникальным идентификатором сообщения ("unique-id listing"). При отсутствии аргумента аналогичная информация выдается для каждого из сообщений в почтовом ящике. Уникальный идентификатор сообщения состоит из 1-70 символов в диапазоне от 0x21 до 0x7E. Сообщения в почтовом ящике должны характеризоваться различными идентификаторами. Пример использования команды:

К: UIDL

С: +OK

С: 1 whqtswO00WBw418f9t5JxYwZ

С: 2 QhdPYR:00WBw1Ph7x7

USER name, где name - характеризует почтовый ящик сервера.

Команда используется на фазе авторизации или после неудачного завершения команд USER или PASS. При авторизации клиент должен сначала послать команду USER и лишь после получения положительного отклика команду PASS. Команда может вызвать следующие отклики:

+OK name is a valid mailbox

-ERR never heard of mailbox name

Примеры использования команды USER:

К: USER frated

С: -ERR sorry, no mailbox for frated here

...

К: USER mrose

С: +OK mrose is a real hoopy frood

PASS string (string - пароль для доступа к почтовому серверу)

Команда работает в режиме авторизации сразу после команды USER. Когда клиент выдает команду PASS, сервер использует аргументы команд USER и PASS для определения доступа клиента к почтовому ящику. На команду PASS возможны следующие отклики:

+OK maildrop locked and ready

-ERR invalid password

-ERR unable to lock maildrop

Пример диалога при использовании команды PASS:

К: USER mrose

С: +OK mrose is a real hoopy frood

К: PASS secret

С: -ERR maildrop already locked

...

К: USER mrose

С: +OK mrose is a real hoopy frood

K: PASS secret

C: +OK mrose's maildrop has 2 messages (320 octets)

APOP name digest, где name - идентификатор почтового ящика, a digest - дайджест сообщения - MD5 (RFC-1828). Команда используется только на стадии авторизации.

Обычно любая сессия начинается с обмена USER/PASS. Но так как в некоторых случаях подключения к серверу POP3 может осуществляться достаточно часто, возрастает риск перехвата пароля. Альтернативным методом авторизации является использование команды APOP. Сервер, который поддерживает применение команды APOP, добавляет временную метку в свое стартовое уведомление. Синтаксис временной метки соответствует формату идентификаторов сообщений, описанному в [RFC822] и должны быть уникальными для всех заголовков уведомлений.

где `process-ID' представляет собой десятичное значение PID процесса, clock - десятичное показание системных часов, a hostname - полное имя домена, где размещен сервер POP3.

Клиент POP3 фиксирует временную метку и выдает команду APOP. Параметр `name' семантически идентичен параметру `name' команды USER. Параметр `digest' вычисляется с использованием алгоритма MD5 [RFC1321] для строки, состоящей из временной метки (включая угловые скобки) за которой следует строка пароля, которая известна только клиенту и серверу. Параметр `digest' содержит 16 октетов, которые пересылаются в шестнадцатеричном формате с использованием строчных ASCII символов. Сервер, получив команду APOP, проверяет принятый дайджест и если он корректен, посылает положительный отклик клиенту. Сессия при этом переходит в состояние транзакции. В противном случае посылается отрицательный отклик и состояние сессии не изменяется. С целью обеспечения безопасности для каждого конкретного пользователя и сервера должен использоваться либо метод доступа

USER/PASS, либо APOP, но не в коем случае оба метода попеременно.

Сервер перед закрытием канала по команде QUIT должен удалить из почтового ящика все сообщения, которые были перенесены с помощью команд RETR.

Предполагается, что все сообщения, передаваемые в ходе сессии POP3, имеют текстовый формат Интернет в соответствии с документом [RFC822].

Протокол Интернет для работы с сообщениями IMAP

Протокол IMAP 4.1 (INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1, V.Crispin, RFC-2060, December 1996) базируется на транспортном протоколе TCP и использует порт 143.

Работает только с сообщениями и не требует каких-либо пакетов со специальными заголовками.

В отличие от POP3 IMAP хранит почтовые сообщения у себя “вечно”.

Атрибуты сообщения

Клиент посылает серверу команды, сервер клиенту данные и уведомления о статусе выполнения запроса.

Все сообщения, как клиента, так и сервера имеют форму строк, которые завершаются последовательностью CRLF.

Каждое сообщение имеет несколько связанных с ним атрибутов.

Доступ к сообщениям в IMAP 4.1 осуществляется с помощью уникального идентификатора или порядкового номера сообщения.

Атрибут сообщения UID

Каждому сообщению ставится в соответствие 32-битовый код, который при использовании совместно с уникальным идентификатором образует 64-битовую последовательность, гарантирующую однозначную идентификацию

сообщения в почтовом ящике.

Хорошим значением UID можно считать 32-битное представление даты и времени создания почтового ящика.

UID сообщения не должно изменяться в пределах сессии, его не следует изменять и от сессии к сессии.

Атрибут порядкового номера сообщения

Этот атрибут определяет порядковый номер сообщения в почтовом ящике, начиная с 1.

При удалении сообщения номера всех последующих сообщений переупорядочиваются.

Атрибут флагов сообщения

Представляет собой список из нуля или более именованных лексем, соотнесенный данному сообщению.

Флаг устанавливается путем его добавления к этому списку и обнуляется путем его удаления.

Определены ***следующие системные флаги:***

seen - Сообщение прочитано;

answered - На сообщение послан ответ;

flagged - Сообщение "помечено" как срочное, требующее особого внимания;

deleted - Сообщение помечено как стертое для последующего удаления посредством expunge;

draft - Сообщения не является законченным (помечено, как проект);

recent - Сообщение только что положено в почтовый ящик.

Флаг не может быть изменен клиентом.

- ***Атрибут «Внутренняя дата и время сообщения на сервере»*** (время и дата получения сообщения).

- ***Атрибут размера сообщения***

определяет число октетов в сообщении.

- ***Атрибут структуры конверта сообщения*** соответствует требованиям документа [RFC-822].

- ***Атрибут структуры тела сообщения*** **несет в себе информацию о структуре сообщения** в соответствии с регламентациями [MIME-IMB].

Формат данных

IMAP 4.1 использует текстовые команды и отклики. *Данные в IMAP 4.1 могут иметь одну из следующих форм:*

- *атом,*

- *число,*

- *строка,*

- *список, заключенный в скобки или NIL.*

Атом состоит из одного или более неспециализированных символов.

Строка может иметь одну из двух форм: литерал или строка в кавычках. *Литеральная форма является основной формой строки. Строка в кавычках является альтернативной формой*, исключающей избыточность литеральной формы за счет ограничений, налагаемых на символы, используемые в строке.

Литерал представляет собой нуль или более октетов (включая CR и LF). Литерал начинается с октета, где хранится число символов. Этот октет заключается в фигурные скобки, за которыми следует последовательность CRLF.

В случае передачи литералов от сервера к клиенту за CRLF следуют непосредственно данные. При передаче литералов от клиента серверу клиент должен подождать прихода команды продолжения, прежде чем начать пересылку данных.

Строка в кавычках представляет собой последовательность из нуля или более 7-битовых символов за исключением CR и LF, начинающуюся и завершающуюся двойной кавычкой (<">). Пустая строка представляется как "" или как литерал {0}, за которым следует последовательность CRLF.

Специальный атом "NIL" представляет собой указание на отсутствие каких-то определенных данных типа строка или "список в скобках".

Его следует отличать от пустой строки "" или пустого "списка в скобках" ().

Команды клиента

Ниже описаны команды IMAP 4.1. *Команды рассматриваются с учетом состояния, в котором они допустимы.*

Следующие команды могут использоваться в любом состоянии:

- CAPABILITY,

- NOOP,

- LOGOUT.

CAPABILITY - запрашивает перечень возможностей, поддерживаемых сервером;

NOOP - ничего не делает и всегда успешно завершается.

LOGOUT - информирует сервер о том, что клиент прерывает соединение.

Команды клиента - в состоянии без аутентификации:

AUTHENTICATE или LOGIN организуют аутентификацию и переводят

систему в состояние с аутентификацией.

AUTHENTICATE - указывает серверу на механизм аутентификации. Если сервер поддерживает запрошенный механизм аутентификации, он выполняет обмен согласно аутентификационному протоколу и идентифицирует клиента.

LOGIN - идентифицирует клиента серверу и передает пароль пользователя открытым текстом.

Команды клиента в состоянии "аутентификация осуществлена":

- *SELECT,* - *EXAMINE,*

- *CREATE,* - *DELETE,*

- *RENAME,* - *SUBSCRIBE,*

- *UNSUBSCRIBE,* - *LIST,*

- *LSUB,* - *STATUS*

-APPEND.

SELECT - осуществляет выбор почтового ящика, так чтобы обеспечить доступ к сообщениям, находящимся там.

EXAMINE - идентична команде SELECT и дает тот же результат, но почтовый ящик идентифицируется как "только для чтения".

CREATE - создает почтовый ящик с заданным именем.

DELETE - навечно удаляет почтовый ящик с указанным именем.

RENAME - изменяет имя почтового ящика.

SUBSCRIBE - добавляет специфицированное имя почтового ящика к списку "активных" или "подписных" ящиков сервера.

UNSUBSCRIBE - удаляет специфицированный почтовый ящик из списка "активных" или "подписных" почтовых ящиков данного сервера.

LIST - возвращает субнабор имен из полного набора.

LSUB - возвращает субнабор имен из списка пользователя, который декларирован как "активный" или "подписной".

STATUS - запрашивает статусные данные для указанного почтового ящика.

APPEND - добавляет литеральный аргумент в качестве нового сообщения в почтовый ящик.

Команды клиента в состоянии "выбор сделан"

В состоянии "выбор сделан", разрешены команды, которые манипулируют сообщениями в почтовом ящике. Помимо универсальных команд (CAPABILITY, NOOP и LOGOUT), а также команд режима аутентификации (SELECT, EXAMINE, CREATE, DELETE, RENAME, SUBSCRIBE, UNSUBSCRIBE, LIST, LSUB, STATUS и APPEND), в данном режиме доступны следующие команды:

CHECK, CLOSE, EXPUNGE, SEARCH, FETCH, STORE, COPY и UID.

CHECK - осуществляет проверку выбранного почтового ящика.

CLOSE - навечно удаляет из выбранного почтового ящика все сообщения, помеченные флагом \Deleted, и возвращает систему в состояние "аутентификация выполнена".

EXPUNGE - навечно удаляет из выбранного почтового ящика все сообщения, которые помечены флагами \Deleted.

SEARCH ищет почтовый ящик, который отвечает выбранным критериям отбора. Критерий отбора состоит из одного или более ключей поиска.

FETCH - извлекает данные, соответствующие сообщению в почтовом ящике.

STORE - заносит данные в почтовый ящик.

COPY - копирует специфицированное сообщение в конец указанного почтового ящика.

UID – предназначена для возврата информации о уникальных идентификаторах или порядковых номерах сообщений.

Команды клиента - экспериментальные и расширения

Любая команда с префиксом X является экспериментальной командой.

Любой добавленный немаркированный отклик, посланный экспериментальной командой также должен иметь префикс X.

Отклики сервера – отклики состояния

Статусными откликами являются:

-OK,

- NO,

- BAD,

- PREAUTH;

- BYE.

OK, NO и BAD могут быть маркированными или нет. PREAUTH и BYE - всегда не маркированы.

Статусные отклики могут включать опционный код отклика.

Код отклика состоит из информации, заключенной в квадратные скобки, в форме атома. Далее может следовать пробел и аргументы.

Код отклика содержит дополнительную информацию или статусные коды для программы клиента помимо условий OK/NO/BAD.

OK - индицирует информационное сообщение от сервера. **Если оно маркировано, сообщение указывает на успешное завершение соответствующей команды.** Пользователю может быть предложено текстовое информационное сообщение. Немаркированная форма указывает на чисто информационное сообщение; природа информации может быть указана в коде отклика.

Немаркированная форма используется также как один из трех видов оповещения об установлении начального соединения. Эта форма указывает, что еще не выполнена аутентификация и необходима команда LOGIN.

***PREAUTH* - является всегда непомеченным и представляет собой одну из трех возможных реакций при установлении соединения.** Он указывает, что для соединения уже выполнена аутентификация, и команда LOGIN не нужна.

BYE - является всегда непомеченным, он указывает, что сервер намеривается разорвать соединение. При этом пользователю может быть послано текстовое сообщение, проясняющее статус клиента.

Отклик BYE посылается при выполнении одного из четырех условий:

- **Как часть нормальной процедуры logout.** Сервер закроет соединение после отправки маркированного отклика OK на команду LOGOUT.

- **Как уведомление об аварийном прерывании сессии.** Сервер немедленно разрывает соединение.

- **Как уведомление о процедуре автоматического logout по причине отсутствия активности.** Сервер немедленно разрывает соединение.

- **Как одно из трех возможных сообщений при установлении соединения, уведомляя, что сервер не может установить соединение с данным клиентом.** Сервер немедленно разрывает соединение.

CAPABILITY - возникает в результате исполнения одноименной команды.

LIST - посылается как результат команды LIST. Он возвращает одно имя, которое соответствует спецификации LIST. Допускается несколько откликов LIST на одну команду.

LSUB - является результатом команды LSUB.

STATUS - является результатом команды STATUS. Он возвращает имя почтового ящика, которое соответствует спецификации STATUS, и запрашиваемую статусную информацию почтового ящика.

RECENT - сообщает число сообщений с флагами \Recent.

Отклики сервера - статусное сообщение

Отклики сервера - статусное сообщение

***EXPUNGE* - уведомляет, сообщение с каким порядковым номером удалено из почтового ящика.** Порядковые номера последующих сообщений немедленно уменьшаются на единицу.

***FETCH* - возвращает данные о сообщении клиенту.** Данные сформированы в группы из имени элемента и его значения в скобках.

Отклики сервера - запрос продолжения команды

Отклики сервера - запрос продолжения команды

Отклик на запрос продолжения команды вместо метки выделяется символом "+". Эта форма отклика указывает на то, что сервер готов принять продолжение команды от клиента. Остальная часть этого отклика имеет текстовую форму.

Этот отклик используется командой AUTHORIZATION, для того чтобы передать данные от сервера клиенту, и запросить дополнительные данные у клиента.

Пример соединения IMAP 4.1

Последующее является записью для соединения IMAP 4.1. (Данный пример и нижеприведенное описание синтаксиса практически без изменения взято из документа RFC -2060).

S: * OK IMAP4rev1 Service Ready

C: a001 login mrc secret

S: a001 OK LOGIN completed

C: a002 select inbox

S: * 18 EXISTS

S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)

S: * 2 RECENT

S: * OK [UNSEEN 17] Message 17 is the first unseen message

S: * OK [UIDVALIDITY 3857529045] UIDs valid

S: a002 OK [READ-WRITE] SELECT completed

C: a003 fetch 12 full

S: * 12 FETCH (FLAGS (\Seen) INTERNALDATE "17-Jul-1996 02:44:25 -0700"

RFC822.SIZE 4286 ENVELOPE ("Wed, 17 Jul 1996 02:23:25 -0700 (PDT)"

"IMAP4rev1 WG mtg summary and minutes"

(("Terry Gray" NIL "gray" "cac.washington.edu"))

(("Terry Gray" NIL "gray" "cac.washington.edu"))

(("Terry Gray" NIL "gray" "cac.washington.edu"))

((NIL NIL "imap" "cac.washington.edu"))

((NIL NIL "minutes" "CNRI.Reston.VA.US")

("John Klensin" NIL "KLENSIN" "INFOODS.MIT.EDU")) NIL NIL

"")

BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028 92))

S: a003 OK FETCH completed

C: a004 fetch 12 body[header]

S: * 12 FETCH (BODY[HEADER] {350}

S: Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)

Криптографические средства защиты

Криптография используется для достижения следующих целей.

- Конфиденциальность: защита данных или личной информации пользователя от несанкционированного просмотра.
- Целостность данных: защита данных от несанкционированного изменения.
- Проверка подлинности: проверка того, что данные исходят действительно от определенного лица.
- Неотрекаемость. Ни одна сторона не должна иметь возможность отрицать факт отправки сообщения.

Для достижения указанных целей можно использовать алгоритмы и правила, известные как криптографические примитивы, для создания криптографической схемы.

Шифрование с закрытым ключом (симметричное шифрование) - Осуществляет преобразование данных с целью предотвращения их просмотра

третьей стороной. В данном способе шифрования для шифрования и расшифровки данных используется один общий закрытый ключ.

Шифрование с открытым ключом (асимметричное шифрование) - Осуществляет преобразование данных с целью предотвращения их просмотра третьей стороной. В данном способе шифрования для шифрования и расшифровки используется набор, состоящий из открытого и закрытого ключей.

Создание криптографической подписи - Позволяет проверить, что данные действительно исходят от конкретного лица, используя для этого уникальную цифровую подпись этого лица. Данный процесс также использует хэш-функции.

Криптографическое хэширование - Отображает данные любого размера в байтовую последовательность фиксированной длины. Результаты хэширования статистически уникальны; отличающаяся хотя бы одним байтом последовательность не будет преобразована в то же самое значение.

Симметричное шифрование

При шифровании с закрытым ключом для шифровки и дешифровки данных используется один закрытый ключ. Необходимо обезопасить этот ключ от несанкционированного доступа, потому что любое обладающее им лицо может использовать его для расшифровки данных или шифрования собственных данных с подменой источника.

Шифрование с закрытым ключом называют также симметричным шифрованием, так как для шифрования и расшифровки используется один и тот же ключ. Алгоритмы шифрования с закрытым ключом являются очень быстрыми (по сравнению с алгоритмами шифрования с открытым ключом) и хорошо подходят для осуществления криптографических преобразований больших массивов информации. Асимметричные алгоритмы шифрования, такие как RSA, имеют математические ограничения на объем шифруемых данных. Для симметричных алгоритмов шифрования подобные проблемы

обычно не возникают.

Разновидность алгоритмов шифрования с закрытым ключом, называемая блочным шифром, используется для шифрования целого блока данных за один раз. Блочные шифры (такие как DES, TripleDES и AES) преобразуют входной блок данных длиной в n байтов в выходной блок зашифрованных данных. Если необходимо зашифровать или расшифровать последовательность байтов, следует делать это блок за блоком. Поскольку n достаточно мало (8 байт для DES и TripleDES; 16 байт [по умолчанию], 24 или 32 байта для AES), данные большей длины, чем n, должны шифроваться блоками, один блок за раз. Блоки данных размером менее n байт должны быть увеличены до n байт перед обработкой.

Одна из простейших форм блочного шифра называется режимом электронной кодовой книги (ECB). Режим ECB не считается безопасным, поскольку в нем не используется вектор инициализации для инициализации первого текстового блока. Для заданного закрытого ключа k простой блочный шифр, не использующий вектор инициализации, зашифрует одинаковые входные блоки текста в одинаковые выходные блоки зашифрованного текста. Поэтому если во входном текстовом потоке присутствуют одинаковые блоки, в зашифрованном потоке также будут присутствовать одинаковые блоки. Такие повторяющиеся выходные блоки сообщают неправомочным пользователям о ненадежных алгоритмах шифрования, которыми можно воспользоваться, о и возможных типах атак. Шифр ECB поэтому очень уязвим для анализа и, в конечном итоге, взлому ключа.

Классы блочных шифров, предоставляемые библиотекой базовых классов, используют режим сцепления, называемый сцеплением шифровальных блоков (CBC), хотя данную настройку по умолчанию можно изменить.

Шифры CBC решают проблемы, связанные с использованием шифров ECB, благодаря использованию вектора инициализации (IV) для шифрования первого текстового блока. Перед шифрованием каждого блока открытого текста

он объединяется с зашифрованным текстом предыдущего блока с помощью поразрядной исключающей операции OR (XOR). Поэтому каждый блок зашифрованного текста зависит от всех предыдущих блоков. При использовании этой системы шифрования стандартные заголовки сообщений, которые могут быть известны неправомочному пользователю, не могут быть использованы им для восстановления закрытого ключа.

Одним из способов раскрытия данных, зашифрованных с помощью шифра CBC, является подбор ключа методом полного перебора возможных ключей. В зависимости от длины использованного для шифрования ключа, такой подбор занимает очень много времени даже на самых быстрых компьютерах, поэтому он является практически неосуществимым. Шифры с большей длиной ключа труднее взломать. Несмотря на то, что использование шифрования не гарантирует теоретическую невозможность раскрытия зашифрованных данных неправомочным лицом, стоимость такого взлома становится чрезвычайно высокой. Если процесс подбора ключа и раскрытия зашифрованных данных занимает три месяца, в то время как сами данные актуальны в течение всего нескольких дней, подбор ключа методом полного перебора не представляет практической ценности.

Недостатком шифрования с закрытым ключом является необходимость того, чтобы две стороны согласовали ключ и вектор инициализации, для чего может потребоваться их передача через систему связи. Вектор инициализации не считается секретным и может передаваться вместе с сообщением в текстовом формате. Однако ключ не должен стать известен неправомочным пользователям. Из-за указанных проблем шифрование с закрытым ключом часто используется в сочетании с шифрованием с открытым ключом для безопасной передачи ключа и вектора инициализации.

Предположим, что Алиса и Боб являются двумя сторонами, которые хотят осуществлять связь по незащищенному каналу; они могли бы воспользоваться шифрованием с закрытым ключом следующим образом. Алиса и Боб

соглашаются использовать некоторый определенный алгоритм (например, AES) с определенным ключом и вектором инициализации. Алиса пишет сообщение и создает сетевой поток, через который можно отправить это сообщение (например, именованный канал или канал электронной почты). Затем она шифрует текст с помощью ключа и вектора инициализации и по интрасети пересылает зашифрованное сообщение и вектор инициализации Бобу. Боб принимает зашифрованный текст и осуществляет расшифровку, используя ранее согласованные ключ и вектор инициализации. Если происходит перехват передаваемых данных, злоумышленник не может восстановить исходное сообщение, так как ему не известны ключ и вектор инициализации. В этой ситуации в секрете должен сохраняться только ключ. В более реалистичном случае либо Алиса, либо Боб создает закрытый ключ и использует шифрование с открытым ключом (асимметричное) для передачи другой стороне закрытого (симметричного) ключа. Дополнительные сведения о шифровании с открытым ключом см. в следующем разделе.

.NET Framework предоставляет следующие классы, которые реализуют алгоритмы шифрования с закрытым ключом:

- AesManaged (введен в .NET Framework 3.5).
- DESCryptoServiceProvider .
- HMACSHA1 (Технически это алгоритм с закрытым ключом, поскольку он представляет собой код проверки подлинности сообщений, вычисляемый на основе хэш-функции шифрования и закрытого ключа. См. подраздел Хэш-коды, приведенный ниже.
- RC2CryptoServiceProvider .
- RijndaelManaged .
- TripleDESCryptoServiceProvider .

Шифрование с открытым ключом

При шифровании с открытым ключом используются закрытый ключ, который должен храниться в секрете от неправомочных пользователей, а также открытый ключ, который может предоставляться кому угодно. Открытый и закрытый ключи математически взаимосвязаны; данные, зашифрованные с помощью открытого ключа, можно расшифровать исключительно с помощью соответствующего закрытого ключа, а цифровая подпись данных, подписанных с помощью закрытого ключа, может быть проверена только с помощью соответствующего открытого ключа. Открытый ключ может быть предоставлен любому лицу; он используется для шифрования данных, которые должны быть отправлены хранителю закрытого ключа. Алгоритмы шифрования с открытым ключом также известны как асимметричные алгоритмы, потому что для шифрования данных требуется один ключ, а для расшифровки — другой ключ. Основное правило шифрования запрещает повторное использование ключа; оба ключа в каждом сеансе шифрования должны быть уникальными. Однако на практике ассиметричные ключи обычно используются подолгу.

Две стороны (Алиса и Боб) могут использовать шифрование с открытым ключом следующим образом. Сначала Алиса создает набор, состоящий из открытого и закрытого ключей. Если Боб хочет послать Алисе зашифрованное сообщение, он запрашивает у Алисы ее открытый ключ. Алиса высылает Бобу свой открытый ключ через незащищенную сеть, и Боб использует полученный ключ для шифрования своего сообщения. Боб пересылает зашифрованное сообщение Алисе, а она расшифровывает полученные данные с помощью своего закрытого ключа. Если Боб получил ключ Алисы по незащищенному каналу, например через открытую сеть, он оказывается уязвимым для атак типа "злоумышленник в середине". Поэтому Бобу необходимо убедиться, что Алиса имеет правильную копию открытого ключа.

Во время передачи Бобу открытого ключа Алисы может произойти несанкционированный перехват ключа третьим лицом. Более того, возможна

ситуация, что то же самое лицо перехватит зашифрованное сообщение Боба. Тем не менее неправомочная сторона не может осуществить расшифровку сообщения с помощью открытого ключа. Сообщение можно расшифровать только с помощью закрытого ключа Алисы, который не передавался через какие-либо системы связи. Алиса не использует свой закрытый ключ для шифрования ответного сообщения Бобу, потому что любое лицо может расшифровать это сообщение с помощью открытого ключа. Если Алиса желает послать Бобу ответное сообщение, она запрашивает у Боба его открытый ключ и шифрует свое сообщение, используя полученный открытый ключ. Затем Боб осуществляет расшифровку сообщения с помощью своего соответствующего закрытого ключа.

В этом случае Алиса и Боб используют шифрование с открытым ключом (асимметричное) для передачи закрытого (симметричного) ключа, а затем пользуются шифрованием с этим закрытым ключом в течение сеанса связи.

В следующем списке сравниваются алгоритмы шифрования с открытым и закрытым ключом.

- Алгоритмы шифрования с открытым ключом используют буфер фиксированного размера, в то время как алгоритмы шифрования с закрытым ключом могут использовать буфер переменного размера.
- Алгоритмы шифрования с открытым ключом не могут быть использованы для сцепления блоков данных в потоки тем же образом, как в алгоритмах шифрования с закрытым ключом, потому что можно шифровать только маленькие порции данных. Соответственно, асимметричные операции не используют такую же потоковую модель, как симметричные операции.
- Шифрование с открытым ключом имеет намного большее пространство ключей (диапазон возможных значений ключа), чем шифрование с закрытым ключом. Поэтому такое шифрование менее уязвимо к атакам полного перебора, когда проверяются все возможные значения ключа.

- Открытые ключи можно легко распространять, поскольку они не требуют особой защиты при условии, что существует некий способ установления подлинности источника.
- Некоторые алгоритмы шифрования с открытым ключом (такие как RSA и DSA, но не Diffie-Hellman) могут быть использованы для создания цифровых подписей, служащих для подтверждения идентичности лица, от которого исходят данные.
- Алгоритмы шифрования с открытым ключом являются весьма медленными (по сравнению с алгоритмами шифрования с закрытым ключом) и не предназначены для шифрования больших объемов данных. Использование шифрования с открытым ключом имеет смысл только при передаче очень малых массивов данных. Обычно шифрование с открытым ключом применяется для того, чтобы зашифровать ключ и вектор инициализации, которые будут использоваться при шифровании с закрытым ключом. После передачи ключа и вектора инициализации используется уже шифрование с закрытым ключом.

.NET Framework предоставляет следующие классы, которые реализуют алгоритмы шифрования с открытым ключом:

- DSACryptoServiceProvider
- RSACryptoServiceProvider
- ECDiffieHellman (базовый класс)
- ECDiffieHellmanCng
- ECDiffieHellmanCngPublicKey (базовый класс)
- ECDiffieHellmanKeyDerivationFunction (базовый класс)
- ECDsaCng

RSA допускает как шифрование, так и подписывание, в то время как DSA может использоваться только для подписывания, а Diffie-Hellman — только для

генерации ключей. В целом, алгоритмы с открытым ключом имеют более ограниченную сферу применения, чем алгоритмы с закрытым ключом.

Цифровые подписи

Алгоритмы шифрования с открытым ключом могут также использоваться для создания цифровых подписей. Цифровые подписи удостоверяют подлинность источника данных (если вы доверяете открытому ключу источника) и защищают целостность данных. С помощью открытого ключа, созданного Алисой, получатель данных, отправленных Алисой, может проверить, что их источником действительно является Алиса, сравнив цифровую подпись полученных данных с открытым ключом Алисы.

Чтобы использовать шифрование с открытым ключом для создания цифровой подписи сообщения, Алиса сначала применяет к этому сообщению алгоритм хэширования, который создает хэш-функцию сообщения. Хэш-функция сообщения является компактным и уникальным отображением данных. Для создания своей личной цифровой подписи Алиса шифрует хэш-функцию сообщения с помощью своего закрытого ключа. При получении сообщения и цифровой подписи Боб расшифровывает подпись с помощью открытого ключа Алисы, восстанавливая зашифрованную хэш-функцию сообщения, а затем осуществляет хэширование сообщения с помощью того же алгоритма, который использовала Алиса. Если вычисленная Бобом хэш-функция в точности совпадает с хэш-функцией, полученной от Алисы, Боб может быть уверен, что сообщение пришло действительно от владельца закрытого ключа, а также в том, что данные в сообщении не подвергались модификациям. Если Боб уверен, что именно Алиса владеет закрытым ключом, он удостоверяется, что сообщение пришло от Алисы.

Примечание

Цифровая подпись может быть проверена любым лицом, потому что открытый ключ источника данных является общедоступным и обычно включается в

формат цифровой подписи. Рассматриваемый метод не обеспечивает секретности сообщения; для обеспечения секретности его следует еще и зашифровать.

.NET Framework предоставляет следующие классы, реализующие алгоритмы создания цифровой подписи.

- DSACryptoServiceProvider
- RSACryptoServiceProvider
- ECDsa (базовый класс)
- ECDsaCng

К началу

Значения хэша

Алгоритмы хэширования преобразуют двоичные последовательности произвольной длины в двоичные последовательности фиксированного меньшего размера, известные как хэш-коды. Хэш-код является числовым представлением порции данных. Если осуществляется хэширование абзаца текста и в нем изменяется хотя бы одна буква, результат хэширования изменится. Если хэш является криптостойким, его код значительно изменится. Например, если изменяется один бит сообщения, результат выполнения криптостойкой хэш-функции может отличаться на 50%. Несколько входных значений могут преобразовываться в один хэш-код. Однако в вычислительном плане немыслимо найти два разных входных набора данных, результаты хэширования которых полностью совпадают.

Две стороны (Алиса и Боб) могут использовать хэш-функцию для проверки целостности сообщений. Им нужно выбрать хэш-алгоритм для подписывания сообщений. Алиса будет писать сообщение, а затем создавать хэш этого сообщения с помощью выбранного алгоритма. Затем стороны могут применять один из следующих вариантов дальнейших действий.

- Алиса отправляет Бобу сообщение с открытым текстом и хэш сообщения (цифровую подпись). Боб получает сообщение, применяет к нему хэш-алгоритм и сравнивает свое значение хэша со значением, полученным от Алисы. Если хэш-коды совпадают, значит, данные не изменялись. Если хэш-коды не совпадают, значит, данные изменялись после создания.

  К сожалению, этот способ не позволяет установить подлинность отправителя. Любой человек может выдавать себя за Алису и отправлять сообщения Бобу. Для этого достаточно подписывать сообщения с помощью такого же хэш-алгоритма, и Боб сможет лишь проверить, что сообщение соответствует подписи. Это одна из форм атаки "злоумышленник в середине". Дополнительные сведения см. в разделе CNG Secure Communication Example.

- Алиса отправляет Бобу сообщение с открытым текстом по незащищенному открытому каналу. По защищенному закрытому каналу она оправляет Бобу хэш сообщения. Боб получает сообщение с открытым текстом, хэширует его и сравнивает хэш со значением, полученным по закрытому каналу. Если хэши совпадают, Боб может сделать два вывода:

  - сообщение не было изменено;
  - отправитель сообщения подлинный (Алиса).

  Для того чтобы такая система работала, Алиса должна скрывать оригинальный хэш-код от всех, кроме Боба.

- Алиса по незащищенному открытому каналу отправляет Бобу сообщение в виде открытого текста, а хэш сообщения помещает на свой общедоступный веб-узел.

  Такой подход позволяет защититься от изменения хэша третьей стороной. Хотя сообщение и его хэш могут видеть все, изменить хэш может только Алиса. Злоумышленнику, который хочет выдать себя за Алису, потребуется доступ к веб-узлу Алисы.

Ни один из описанных выше способов не защищает Алису от чтения ее сообщений третьими лицами, поскольку сообщения передаются в виде открытого текста. Для обеспечения полной защиты обычно требуется использовать цифровые подписи и шифрование.

Платформа .NET Framework предоставляет следующие классы, реализующие алгоритмы хэширования.

- HMACSHA1 .
- MACTripleDES .
- MD5CryptoServiceProvider .
- RIPEMD160 .
- SHA1Managed .
- SHA256Managed .
- SHA384Managed .
- SHA512Managed .
- Разновидности HMAC всех алгоритмов SHA, MD5 и RIPEMD-160.
- Реализации CryptoServiceProvider (оболочки управляемого кода) всех алгоритмов SHA.
- Реализации криптографии следующего поколения (CNG) всех алгоритмов MD5 и SHA.

Примечание

Недостатки алгоритма MD5 были выявлены еще в 1996 году, и вместо него был рекомендован алгоритм SHA-1. В 2004 году были обнаружены дополнительные уязвимости, и в настоящее время алгоритм MD5 уже не считается безопасным. Алгоритм SHA-1 также был признан ненадежным, и вместо него рекомендуется использовать SHA-2.

Генерация случайных чисел

Процесс генерации случайных чисел является составной частью многих криптографических операций. Например, криптографические ключи должны выбираться настолько случайно, насколько это возможно, чтобы было фактически невозможно воспроизвести их значения. Криптографические генераторы случайных чисел должны генерировать результат, который нельзя предсказать вычислительными методами с вероятностью хотя бы 50%. Поэтому все методы предсказания очередного случайного бита не должны быть точнее, чем простое угадывание. Классы .NET Framework используют генераторы случайных чисел для создания криптографических ключей.

Класс RNGCryptoServiceProvider является реализацией алгоритма генерации случайных чисел.

Манифест ClickOnce

В .NET Framework 3.5 следующие криптографические классы используются проверки и получения сведений о подписях манифестов для приложений, которые развертываются с применением технологии ClickOnce.

- Класс ManifestSignatureInformation получает данные о подписи манифеста при использовании его перегрузок метода VerifySignature.
- Можно использовать перечисление ManifestKinds для указания манифеста для проверки. Результатом проверки является одно из значений перечисления SignatureVerificationResult.
- Класс ManifestSignatureInformationCollection предоставляет коллекцию объектов проверенных подписей ManifestSignatureInformation только для чтения.

Кроме того, следующие классы предоставляют сведения о конкретных подписях:

- StrongNameSignatureInformation содержит сведения о строгом имени

подписи для манифеста.

- AuthenticodeSignatureInformation представляет сведения о подписи Authenticode для манифеста.
- TimestampInformation содержит сведения об отметке времени на подписи Authenticode.
- TrustStatus предоставляет простой способ проверить, является ли подпись Authenticode достоверной.

К началу

Поддержка стандарта Suite B

Платформа .NET Framework 3.5 поддерживает набор криптографических алгоритмов стандарта Suite B, опубликованных национальным агентством по безопасности (NSA). Дополнительные сведения о шифровании Suite B см. в разделе NSA Suite B Cryptography Fact Sheet.

Включены следующие алгоритмы.

- Дополнительный стандарт шифрования Advanced Encryption Standart (AES) с размером ключа для шифрования от 128 до 256 битов.
- Алгоритмы SHA-1, SHA-256, SHA-384 и SHA-512 для хеширования. (Последние три обычно объединяются и известны как SHA-2.)
- Алгоритм цифровой подписи ECDSA, использующий кривые в 256, 384 и 521 битов для подписи. В документации NSA приводится определение этих кривых, которые называются в соответствии с ней P-256, P-384 и P-521. Этот алгоритм предоставляется в классе ECDsaCng. Он позволяет отмечать закрытым ключом и проверять подпись открытым ключом.
- Алгоритм ECDH, использующий кривые в 256, 384 и 521 битов для обмена ключами/тайного соглашения. Этот алгоритм предоставляется в классе ECDiffieHellmanCng.

Оболочки управляемого кода для сертифицированных Федеральным стандартом обработки информации (FIPS) реализаций AES, SHA-256, SHA-384 и SHA-512 доступны в новых классах AesCryptoServiceProvider, SHA256CryptoServiceProvider, SHA384CryptoServiceProvider и SHA512CryptoServiceProvider.

К началу

Классы криптографии следующего поколения (CNG)

Классы криптографии следующего поколения (CNG) предоставляют управляемую оболочку для собственных функций CNG. (CNG является заменой CryptoAPI.) В имена этих классов входит подстрока "Cng". Центральным в группе классов оболочек CNG является класс контейнера ключей CngKey, который абстрагирует хранение и использование CNG ключей. Этот класс позволяет безопасно хранить пару ключей или открытый ключ и ссылаться на него, используя простое строковое имя. Класс цифровых подписей ECDsaCng и класс шифрования ECDiffieHellmanCng на основе эллиптических кривых используют объекты CngKey.

Класс CngKey используется для множества дополнительных операций, включая открытие, создание, удаление и экспорт ключей. Он также предоставляет доступ к базовому ключу дескриптора, используемому при непосредственном вызове собственных функций.

.NET Framework 3.5 также включает множество вспомогательных классов CNG, некоторые из которых приведены ниже.

- CngProvider поддерживает поставщик хранилища ключей.
- CngAlgorithm поддерживает алгоритм CNG.
- CngProperty поддерживает часто используемые свойства ключей.

Для более подробной информации см. продолжение по ссылке:

http://msdn.microsoft.com/ru-ru/library/92f9ye3s%28v=vs.110%29.aspx#secret_key

Задание к курсовому проекту

Требования к разработке почтового клиента:

1) Для отправки почты использовать протокол SMTP.
2) Для получения почты использовать один из протоколов электронной почты: IMAP или POP3.
3) Для безопасной аутентификации использовать защищенные протоколы аутентификации (см. рекомендации администраторов почтовых серверов)
4) Возможность получения почты с вложенными файлами.
5) Возможность чтения писем на русском языке.
6) Форматирование письма.
7) Возможность изменения параметров для подключения к почтовому серверу: название почтового сервера, login, password, порт протокола и т. д. (смотри изменение параметров в стандартных почтовых клиентах).
8) Возможность создания нескольких почтовых ящиков на одном клиенте и переключение между ними.
9) Сохранение писем в ящике (должно быть предусмотрено хранение писем в папке «входящие», «отправленные», «черновики», «корзина» в каждом из настроенных почтовых ящиков).
10) Выполнение синхронизации папок клиента с папками на почтовом сервере.
11) Использование криптографических алгоритмов для шифрования почтового сообщения и использование ЭЦП:

- шифрование тела сообщения симметричным алгоритмом шифрования (см. вариант);

- шифрование ключа симметричного алгоритма ассиметричным алгоритмов (см. вариант);

- получение дайжеста сообщения с помощью функции хеширования (см.вариант)

- реализация ЭЦП с помощью ассиметричного алгоритма (см. вариант).

10) Разработка понятного пользовательского интерфейса.

11) Тестирование почтового клиента в реальных условиях.

12) Самостоятельное изучение библиотеки PGP. Сделать сравнительный анализ средств защиты электронной почты библиотеки PGP и криптографических алгоритмов, которые использовались при реализации клиента.

Задание по использованию криптографических алгоритмов в почтовом клиенте смотри в таблице 6:

Возможно использование других алгоритмов шифрования, не представленных в таблице вариантов (утвердить вариант у преподавателя).

Таблица 6 – Варианты заданий к курсовому проекту

Вариант по журналу	Симметричный.алгоритм шифрования, размер ключа выбрать самостоятельно	Ассиметричный алгоритм шифрования для шифрования ключей симметричного алгоритма	Алгоритм хеширования	Ассиметричный алгоритм шифрования для ЭЦП
1	**AES**	**RSA**	**SHA1**	**DSA**
2	**DES**	**RSA**	**MD5**	**RSA**
3	**TripleDES**	**RSA**	**SHA2**	**Diffie-Hellman**
4	**Rijndael**	**RSA**	**SHA1**	**Diffie-Hellman**
5	**AES**	**RSA**	**MD5**	**RSA**
6	**DES**	**RSA**	**SHA2**	**Diffie-Hellman**
7	**TripleDES**	**RSA**	**SHA1**	**DSA**
8	**Rijndael**	**RSA**	**MD5**	**DSA**
9	**AES**	**RSA**	**SHA2**	**Diffie-Hellman**
10	**DES**	**RSA**	**SHA1**	**DSA**
11	**TripleDES**	**RSA**	**MD5**	**RSA**
12	**Rijndael**	**RSA**	**SHA2**	**RSA**
13	**AES**	**RSA**	**SHA1**	**DSA**
14	**DES**	**RSA**	**MD5**	**RSA**
15	**TripleDES**	**RSA**	**SHA2**	**Diffie-Hellman**
16	**Rijndael**	**RSA**	**SHA1**	**Diffie-Hellman**
17	**AES**	**RSA**	**MD5**	**RSA**
18	**DES**	**RSA**	**SHA2**	**Diffie-Hellman**

19	**TripleDES**	**RSA**	**SHA1**	**DSA**
20	**Rijndael**	**RSA**	**MD5**	**DSA**
21	**AES**	**RSA**	**SHA2**	**Diffie-Hellman**
22	**DES**	**RSA**	**SHA1**	**DSA**
23	**TripleDES**	**RSA**	**MD5**	**RSA**
24	**Rijndael**	**RSA**	**SHA2**	**RSA**
25	**AES**	**RSA**	**MD5**	**DSA**
26	**DES**	**RSA**	**SHA2**	**DSA**
27	**TripleDES**	**RSA**	**SHA1**	**Diffie-Hellman**
28	**Rijndael**	**RSA**	**MD5**	**Diffie-Hellman**
29	**DES**	**RSA**	**SHA2**	**DSA**
30	**TripleDES**	**RSA**	**SHA1**	**DSA**

Содержание пояснительной записки курсового проекта:

1. Анализ существующих почтовых клиентов
2. Проектирование почтового клиента

2.1 Протокол SMTP и его расширения

2.2 Протокол POP3

2.3 Протокол IMAP

2.4 Общая структура приложения.

2.5 Основные требования к программному продукту с учетом требований безопасности.

2.5 UML-диаграммы.

3. Теоретическое описание используемых криптографических алгоритмов в почтовом клиенте

3.1 Симметричный алгоритм шифрования (алгоритм по варианту)\

3.2 Ассиметричный алгоритм шифрования (для шифрования ключей симметричного алгоритма, алгоритм по варианту).

3.3 Описание функции хеширования (однонаправленная хеш-функция по варианту)

3.4 Ассиметричный алгоритм шифрования для ЭЦП (алгоритм по варианту)

4. Разработка почтового клиента

4.1. Выбор средств реализации. Обоснование выбора.

4.4 Описание библиотек, которые использовались для реализации почтового клиента (реализация почтовых протоколов, криптографические библиотеки)

4.2 Описание классов (с учетом криптографических средств защиты)

4.3 Описание методов (с учетом криптографических средств защиты)

4.4 Описание основных алгоритмов работы программы.

4.5 Разработка интерфейса системы

5 Тестирование почтового клиента. Анализ результатов.

6 Руководство пользователя

Приложение А – Техническое задание к курсовому проекту

Приложение Б – Листинг программ

Приложение В – Экранные формы работы программы

Техническое здание оформить по образцу любого ТЗ из предыдущего курсового проекта.