Network Working Group
Request for Comments: 4253
Category: Standards Track

T. Ylonen
SSH Communications Security Corp
C. Lonvick, Ed.
Cisco Systems, Inc.

January 2006

Транспортный протокол SSH

The Secure Shell (SSH) Transport Layer Protocol

Статус документа

В этом документе содержится спецификация протокола, предложенного сообществу Internet. Документ служит приглашением к дискуссии в целях развития и совершенствования протокола. Текущее состояние стандартизации протокола вы можете узнать из документа "Internet Official Protocol Standards" (STD 1). Документ может распространяться без ограничений.

Авторские права

Аннотация

Протокол SSH[1] используется для организации безопасного входа в удалённую систему (login) и организации иных безопасных служб через сети, не обеспечивающие безопасности. В этом документе описан протокол транспортного уровня SSH, который обычно работает на базе TCP/IP. Протокол может использоваться в качестве базы для множества защищённых сетевых служб. Он обеспечивает сильное шифрование, аутентификацию серверов и защиту целостности. Протокол может также обеспечивать компрессию.

Метод обмена ключами, алгоритм открытых ключей, симметричный алгоритм шифрования, алгоритм аутентификации сообщений и алгоритм хэширования согласуются сторонами.

В этом документе также описан метод обмена ключами Diffie-Hellman и минимальный набор алгоритмов, требуемых для реализации транспортного уровня SSH.

Оглавление

1. Введение....2
2. Разработчики....2
3. Используемые в документе соглашения....2
4. Организация соединения....2
4.1. Работа «поверх» TCP/IP....2
4.2. Обмен информацией о версии протокола....2
5. Совместимость со старыми версиями SSH....3
5.1. Старый клиент, новый сервер....3
5.2. Новый клиент, старый сервер....3
5.3. Размер пакетов и дополнительные издержки....3
6. Протокол двоичных пакетов....4
6.1. Максимальный размер пакета....4
6.2. Компрессия....4
6.3. Шифрование....4
6.4. Целостность данных....5
6.5. Методы обмена ключами....6
6.6. Алгоритмы открытых ключей....6
7. Обмен ключами....7
7.1. Согласование алгоритма....7
7.2. Вывод при обмене ключами....9
7.3. Принятие ключей для использования....9
8. Обмен ключами по методу Diffie-Hellman....9
8.1. diffie-hellman-group1-sha1....10
8.2. diffie-hellman-group14-sha1....10
9. Повторный обмен ключами....10
10. Запрос обслуживания....10
11. Дополнительные сообщения....11
11.1. Сообщение о разъединении....11
11.2. Сообщение о проигнорированных данных....11
11.3. Отладочное сообщение....11
11.4. Резервные сообщения....12
12. Номера сообщений....12
13. Согласование с IANA....12
14. Вопросы безопасности....12
15. Литература....12
15.1. Нормативные документы....12
15.2. Дополнительная литература....13
Адреса авторов....13
Торговые марки....13

[1]Secure Shell.

1. Введение

Транспортный уровень SSH обеспечивает защищённый транспорт с сильным шифрованием, криптографической аутентификацией хостов и защитой целостности.

Аутентификация на этом уровне протокола осуществляется для хостов, аутентификации пользователей на транспортном уровне нет. Для аутентификации пользователей может быть разработан протокол вышележащего уровня, работающий на базе данного протокола.

Протокол разработан с учётом простоты и гибкости согласования параметров и минимизации числа циклов кругового обхода при согласовании. Согласуются метод обмена ключами, алгоритм открытых ключей, алгоритм аутентификации сообщений и алгоритм хэширования. Предполагается, что в большинстве сред достаточно будет 2 циклов кругового обхода для полного обмена ключами, аутентификации сервера, запроса обслуживания и восприятия уведомления о запросе сервиса. Максимальное число циклов, которые могут потребоваться для согласования - 3.

2. Разработчики

Основными разработчиками этого комплекта документов являются: Tatu Ylonen, Tero Kivinen, Timo J. Rinne, Sami Lehtinen (все из SSH Communications Security Corp) и Markku-Juhani O. Saarinen (университет Jyvaskyla). Darren Moffat был редактором этого комплекта документов и внёс важный вклад в работу.

За годы подготовки этого документа множество людей внесло свой вклад. В их число входят: Mats Andersson, Ben Harris, Bill Sommerfeld, Brent McClure, Niels Moller, Damien Miller, Derek Fawcus, Frank Cusack, Heikki Nousiainen, Jakob Schlyter, Jeff Van Dyke, Jeffrey Altman, Jeffrey Hutzelman, Jon Bright, Joseph Galbraith, Ken Hornstein, Markus Friedl, Martin Forssen, Nicolas Williams, Niels Provos, Perry Metzger, Peter Gutmann, Simon Josefsson, Simon Tatham, Wei Dai, Denis Bider, der Mouse и Tadayoshi Kohno. Указанные в списке люди могли не участвовать в написании данного документа, но они внесли свой вклад в его подготовку.

3. Используемые в документе соглашения

Во всех документах, связанных с протоколом SSH, следует использовать ключевые слова: **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) для описания уровня требования. Интерпретация этих слов описана в [RFC2119].

Ключевые слова **приватное использование** (PRIVATE USE), **иерархическое выделение** (HIERARCHICAL ALLOCATION), **выделение в соответствии с порядком запросов** (FIRST COME FIRST SERVED), **экспертное рассмотрение** (EXPERT REVIEW), **требуется спецификация** (SPECIFICATION REQUIRED), **одобрение IESG** (IESG APPROVAL), **согласование с IETF** (IETF CONSENSUS), **стандартизация** (STANDARDS ACTION) в данном документе при их использовании в контексте распределения пространства имён интерпретируются в соответствии с [RFC2434].

В данном наборе документов определяются поля протокола и возможные значения этих полей. Поля будут определяться в определениях протокольных сообщений. Например, поле SSH_MSG_CHANNEL_DATA определяется следующим образом.

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel (канал получателя)
string    data (данные)
```

В данном документе поля протокола будут указываться в одинарных кавычках, а значения полей – в двойных. В приведённом выше примере поле data может содержать значения "foo" и "bar".

4. Организация соединения

SSH работает на основе любого 8-битового бинарно-прозрачного транспорта. На нижележащем транспортном уровне **следует** обеспечивать защиту от ошибок при передаче, поскольку такие ошибки вызывают разрыв соединений SSH.

Соединение инициируется со стороны клиента.

4.1. Работа «поверх» TCP/IP

При использовании со стеком протоколов TCP/IP сервер обычно прослушивает порт 22. Этот номер порта зарегистрирован в агентстве IANA и официально выделен для SSH.

4.2. Обмен информацией о версии протокола

Когда соединение организовано, обе стороны **должны** передать идентификационную строку. Эта строка **должна** иметь вид6

```
SSH-protoversion-softwareversion SP comments CR LF
```

Поскольку определяемый в этом наборе документов протокол является версией 2.0, поле protoversion должно содержать значение "2.0". Строка комментария (comments) является **необязательной**. При включении комментария между ним и полем softwareversion должен включаться символ пробела (space, указанный выше, как SP, ASCII 32). Строка идентификации **должна** завершаться одним символом возврата каретки (CR[1]) и одним символом перевода строки (LF[2]), с кодами 13 и 10, соответственно. Разработчики, желающие обеспечить совместимость со старыми, недокументированными версиями этого протокола, могут обрабатывать идентификационные строки без символа возврата каретки по причинам, описанным в разделе 5 этого документа. Передавать null-символ **недопустимо**. Максимальный размер строки идентификации составляет 255 символов с учётом CR и LF.

Часть строки идентификации перед символами CR и LF служит для обмена ключами Diffie-Hellman (см. раздел 8).

[1]Carriage Return.
[2]Line Feed.

Сервер **может** передавать другие строки и данные перед отправкой строки с версией. Каждую строку **следует** завершать парой символов CR и LF. Такие строки **недопустимо** начинать с префикса "SSH-" и **следует** использовать в них кодировку ISO-10646 UTF-8 [RFC3629] (язык не задаётся). Клиенты **должны** быть способны обрабатывать такие строки. Полученная строка **может** игнорироваться или выводиться пользователю на стороне клиента. При выводе строки **следует** использовать фильтрацию управляющих символов, как описано в [SSH-ARCH]. Основным назначением этой функции является обеспечение возможности вывода сообщений об ошибках до разрыва соединения (с помощью TCP-wrapper или иных программ).

Оба поля protoversion и softwareversion должны содержать печатаемые символы US-ASCII за исключением символов пробела и знака «минус» (-). Поле softwareversion используется в основном для инициирования расширений совместимости и индикации возможностей реализации. В поле comments **следует** указывать дополнительную информацию, которая может помочь при решении пользовательских проблем. Пример корректной строки идентификации показан ниже.

```
SSH-2.0-billsSSH_3.6.3q3<CR><LF>
```

Это строка не содержит поля комментариев и завершается символами CR и LF сразу после поля softwareversion.

Обмен ключами начинается сразу после передачи идентификационных строк. В пакетах после строки идентификации **нужно** использовать протокол двоичных пакетов, который описан в разделе 6.

5. Совместимость со старыми версиями SSH

Как отмечено выше поле protoversion для данного протокола имеет значение "2.0". Ранние версии этого протокола не были формально документированы, но широко известно, что в них для поля protoversion используются значения "1.x" (например, "1.5" или "1.3"). На момент написания этого документа многие реализации SSH использовали протокол версии 2.0, но известно, что сохраняются устройства, использующие более ранние версии. В течение переходного периода важно организовать работу так, чтобы обеспечивалась совместимость с имеющимися клиентами и серверами SSH, использующими старые версии протокола. Информация в этом разделе относится только к реализациям, поддерживающим совместимость с SSH версий 1.x. Для интересующихся отметим, что единственной документацией для протоколов версий 1.x являются файлы README, распространяемые вместе с исходными кодами программ [ssh-1.2.30].

5.1. Старый клиент, новый сервер

Серверные реализации **могут** поддерживать конфигурационный флаг совместимости, который позволяет включить совместимость со старыми версиями. При установленном флаге серверу **следует** указывать в поле protoversion значение "1.99". Клиенты, использующие протокол версии 2.0, **должны** быть способны воспринимать этот номер, как идентичный значению "2.0". В этом режиме серверу **следует** передавать символ возврата каретки (CR ASCII 13) после строки идентификации.

В режиме совместимости серверу **не следует** передавать каких-либо дополнительных данных после отправки идентификационной строки, пока не будет получена строка идентификации от клиента. Сервер по этой строке может определить, использует ли клиент старый протокол и, при необходимости вернуться к старому протоколу. В режиме совместимости серверу **недопустимо** передавать какие-либо дополнительные данные перед строкой идентификации.

Когда совместимость со старыми клиентами не требуется, сервер **может** передавать начальные данные обмена ключами сразу после отправки идентификационной строки.

5.2. Новый клиент, старый сервер

Поскольку новый клиент **может** передавать дополнительные данные сразу после идентификационной строки (до получения строки идентификации сервера), работа старого протокола может уже оказаться нарушенной, когда клиент узнает о старом сервере. В таких случаях клиенту **следует** закрыть соединение с сервером и потом подключиться снова, используя старый протокол.

5.3. Размер пакетов и дополнительные издержки

Некоторых читателей может обеспокоить увеличение размера пакетов в результате появления новых заголовков, заполнения и кода аутентификации сообщения (MAC[1]). Минимальный размер пакета составляет примерно 28 байтов (в зависимости от согласованного алгоритма). Это увеличение практически не заметно для больших пакетов, но очень существенно для однобайтовых пакетов (сеансы типа telnet). Однако есть несколько факторов, которые устраняют эту проблему почти во всех случаях:

- минимальный размер заголовка TCP/IP составляет 32 байта; таким образом, размер минимального пакета измениться с 33 до 51 байта (грубо);
- минимальный размер поля данных в кадре Ethernet составляет 46 байтов [RFC0894]; таким образом, размер кадра увеличится не более, чем на 5 байтов; с учётом заголовка Ethernet это увеличение составит менее 10%;
- доля данных типа telnet в сети Internet пренебрежимо мала даже с учётом увеличения размера пакетов.

Единственной средой, где увеличение размера пакетов может приводить к значимым эффектам, является PPP [RFC1661] при работе по медленным модемным каналам (PPP сжимает заголовки TCP/IP, что ведёт к росту относительного увеличения размера пакетов). Однако для современных модемов увеличение времени передачи составит примерно 2 миллисекунды, что значительно быстрее, нежели человек может думать.

Имеются также проблемы, связанные с максимальным размером пакетов. Для минимизации задержки обновления экрана некоторые люди не хотят использовать слишком большие пакеты в интерактивных сеансах. Максимальный размер пакетов согласуется независимо для каждого канала.

[1] Message Authentication Code.

6. Протокол двоичных пакетов

Каждый пакет имеет формат, показанный на рисунке.

```
uint32    packet_length
byte      padding_length
byte[n1]  payload; n1 = packet_length - padding_length - 1
byte[n2]  random padding; n2 = padding_length
byte[m]   mac (Message Authentication Code - MAC); m = mac_length
```

packet_length

Размер пакета в байтах без учёта полей mac и packet_length.

padding_length

Размер случайного заполнения (random padding) в байтах.

payload

Полезное содержимое пакета. Если согласовано использование компрессии, это поле сжимается. Изначально для компрессии **должно** быть установлено значение "none".

random padding

Произвольного размера заполнение, делающие общий размер (packet_length || padding_length || payload || random padding) кратным большему из значений размера блока шифрования или 8 байтов. Размер заполнения **должен** быть не менее 4 байтов. Для заполнения **следует** использовать случайные байты. Максимальный размер заполнения составляет 255 байтов.

mac

Код аутентификации сообщения. Если согласована аутентификация сообщений, это поле содержит байты кода MAC. Изначально к качестве алгоритма MAC **должно** быть задано значение "none".

Отметим, что суммарный размер полей packet_length, padding_length, payload и random padding **должен** быть кратным размеру большему из значений размера блока шифрования или 8 байтов. Это ограничение **должно** выполняться даже при использовании потокового шифра. Отметим, что поле packet_length тоже шифруется и обработка его требует мер предосторожности при передаче и приёме пакетов. Отметим также, что вставка переменного числа байтов заполнения random padding может осложнить анализ трафика.

Минимальный размер пакета равен 16 байтам (или размеру блока шифрования, если тот больше 16) (плюс mac). Реализациям **следует** расшифровывать поле размера после получения первых 8 байтов (или числа байтов в блоке шифрования, если оно больше 8) принимаемого пакета.

6.1. Максимальный размер пакета

Все реализации **должны** быть способны обрабатывать пакеты с размером несжатых данных (payload) до 32768 байтов, а общим размером до 35000 байтов (включая поля packet_length, padding_length, payload, random padding, mac). Максимальное значение 35000 выбрано достаточно произвольно из числа значений, превышающих указанный выше размер несжатой полезной информации. Реализациям **следует** поддерживать более длинные пакеты, когда это может потребоваться. Например, если предполагается, что реализация будет передавать очень большое число сертификатов, можно использовать более крупные пакеты, если идентификационная строка показывает, что другая сторона способна их обработать. Однако реализациям **следует** проверять разумность размера пакетов, чтобы предотвратить атаки на отказ служб или переполнение буферов.

6.2. Компрессия

Если была согласована компрессия, поле payload (и только оно) будет сжиматься с помощью согласованного алгоритма. Значения полей packet_length и mac будут рассчитываться для сжатых данных. Шифрование будет выполняться после компрессии. Компрессия **может** выполняться с учётом состояния, в зависимости от метода. Компрессия **должна** выполняться независимо для каждого направления и реализации **должны** разрешать независимый выбор алгоритма сжатия для каждого направления. На практике, однако, **рекомендуется** использовать общий алгоритм для обоих направлений.

В настоящее время определены следующие алгоритмы компрессии:

```
none     ОБЯЗАТЕЛЬНО     без компрессии
zlib     ОПЦИОНАЛЬНО     сжатие ZLIB (LZ77)
```

Компрессия "zlib" описана в документах [RFC1950] и [RFC1951]. Контекст сжатия инициализируется после каждого обмена ключами и передаётся от пакета к следующему и только частичная очистка выполняется в конце каждого пакета. Частичная очистка подразумевает, что текущий сжатый блок закончился и все данные переданы на выход. Если текущий блок не является сохраненным блоком, после текущего добавляется один или два пустых блока, чтобы гарантировать, что не будет более 8 битов от начала кода завершения (end-of-block) текущего блока и окончанием данных пакета.

Могут быть определены дополнительные алгоритмы компрессии, как указано в [SSH-ARCH] и [SSH-NUMBERS].

6.3. Шифрование

Алгоритм шифрования и ключ будут согласовываться на этапе обмена ключами. Когда шифрование используется, поля размера, данных и заполнения в каждом пакете **должны** шифроваться с использованием данного алгоритма.

Зашифрованные данные по всех пакетах, передаваемых в одном направлении, **следует** рассматривать, как единый поток данных. Например, векторы инициализации **следует** размещать в конце одного пакета и начале следующего. Для всех шифров **следует** использовать ключи размером не менее 128 битов.

Шрифты для каждого из направлений **должны** работать независимо от другого направления. Разработчики должны обеспечивать возможность независимого выбора алгоритма для каждого направления, если локальная политика позволяет использовать множество алгоритмов. Однако на практике **рекомендуется** использовать один алгоритм для обоих направления.

Определённые в настоящее время алгоритмы перечислены в таблице.

Название	*Требование*	*Описание*
3des-cbc	**Требуется**	Трехключевой алгоритм 3DES в режиме CBC[1]
blowfish-cbc	**Не обязательно**	Blowfish в режиме CBC
twofish256-cbc	**Не обязательно**	Twofish в режиме CBC с ключом 256 битов
twofish-cbc	**Не обязательно**	Псевдоним для twofish256-cbc (сохранен в силу традиции)
twofish192-cbc	**Не обязательно**	Twofish с ключом 192 бита
twofish128-cbc	**Не обязательно**	Twofish с ключом 128 битов
aes256-cbc	**Не обязательно**	AES в режиме CBC с ключом 256 битов
aes192-cbc	**Не обязательно**	AES с ключом 192 бита
aes128-cbc	**Рекомендуется**	AES с ключом 128 битов
serpent256-cbc	**Не обязательно**	Serpent в режиме CBC с ключом 256 битов
serpent192-cbc	**Не обязательно**	Serpent с ключом 192 бита
serpent128-cbc	**Не обязательно**	Serpent с ключом 128 битов
arcfour	**Не обязательно**	Потоковый шифр ARCFOUR с ключом 128 битов
idea-cbc	**Не обязательно**	IDEA в режиме CBC
cast128-cbc	**Не обязательно**	CAST-128 в режиме CBC
none	**Не обязательно**	Без шифрования, **не рекомендуется**

Шифр "3des-cbc" представляет собой 3-ключевой вариант triple-DES (шифрование-дешифрование-шифрование), где для первого шифрования используются первые 8 байтов ключа, следующий 8 байтов служат для расшифровки, а последние 8 - для окончательного шифрования. Для этого требуется 24 байта ключевой информации (реально используется 168 битов). Для реализации режима CBC **должна** использоваться цепочка на выходе (т. е., используется только один вектор инициализации). Это блочное шифрование с размером блока 8 байтов. Алгоритм определён в [FIPS-46-3]. Отметим, что в результате того, что эффективный размер ключа в этом алгоритме составляет лишь 112 битов ([SCHNEIER]), алгоритм не соответствует требованию спецификации SSH к алгоритмам шифрования (размер ключа не менее 128 битов). Однако этот алгоритм сохраняется в числе **требуемых** по историческим причинам; все известные на момент написания этого документа реализации поддерживали данный алгоритм и он по-прежнему широко используется по причине интероперабельности. В будущем ожидается появление другого алгоритма (более сильное шифрование), который станет столь повсеместно распространенным, что использование "3des-cbc" будет отменено другим документом (STANDARDS ACTION).

Шифр "blowfish-cbc" представляет собой алгоритм Blowfish в режиме CBC с ключами размером 128 битов [SCHNEIER]. Это блочный шифр с размером блока 8 байтов.

Шифры "twofish-cbc" и "twofish256-cbc" используют алгоритм Twofish в режиме с ключами размером 256 битов, как описано в [TWOFISH]. Это блочный шифр с размером блока 16 байтов.

Шифр "twofish192-cbc" представляет собой вариант описанных выше шифров с размером ключа 192 бита.

Шифр "twofish128-cbc" представляет собой вариант описанных выше шифров с размером ключа 128 битов.

Шифр "aes256-cbc" представляет собой алгоритм AES2 [FIPS-197], в режиме CBC. Эта версия использует 256-битовый ключ.

Шифр "aes192-cbc" представляет собой вариант описанного выше шифра с размером ключа 192 бита.

Шифр "aes128-cbc" представляет собой вариант описанного выше шифра с размером ключа 128 битов.

Шифр "serpent256-cbc" представляет режим CBC с ключом 256 битов для шифра, описанного в Serpent AES.

Шифр "serpent192-cbc" представляет собой вариант описанного выше шифра с размером ключа 192 бита.

Шифр "serpent128-cbc" представляет собой вариант описанного выше шифра с размером ключа 128 битов.

Шифр "arcfour" представляет собой потоковый шифр Arcfour с размером ключа 128 битов. Шифр Arcfour считается совместимым с шифром RC4 [SCHNEIER]. Для шифра Arcfour (и RC4) известна проблема, связанная со слабыми ключами, поэтому его следует применять с осторожностью.

Шифр "idea-cbc" представляет собой алгоритм IDEA в режиме CBC [SCHNEIER].

Шифр "cast128-cbc" представляет собой алгоритм CAST-128 в режиме CBC с размером ключа 128 битов [RFC2144].

Вариант "none" не использует никакого шифрования. Отметим, что этот метод не обеспечивает защиты конфиденциальности и его применение **не рекомендуется**. При выборе этого метода некоторые функции (например, аутентификация по пароля) могут быть отключены из соображений безопасности.

Могут быть определены дополнительные методы шифрования в соответствии с [SSH-ARCH] и [SSH-NUMBERS].

6.4. Целостность данных

Защита целостности данных обеспечивается путём включения в каждый пакет кода MAC, который рассчитывается с использованием разделяемого секрета, порядкового номера пакета и его содержимого.

Алгоритм и ключ аутентификации сообщений согласуются на этапе обмена ключами. Изначально MAC не используется и размер поля **должен** быть равен 0. После обмена ключами значение mac для выбранного алгоритма MAC рассчитывается перед шифрованием для конкатенации данных пакета:

```
mac = MAC(key, sequence_number || unencrypted_packet)
```

[1]Cipher-block chaining - цепочка зашифрованных блоков. *Прим. перев.*

где unencrypted_packet - весь пакет без поля mac (поля размера, payload и random padding), а sequence_number - неявный порядковый номер пакета в формате uint32. Значение sequence_number устанавливается нулевым для первого пакета и увеличивается после каждого пакета (независимо от применения шифрования и MAC). Порядковые номер никогда не сбрасываются (даже при последующей смене ключей или алгоритмов). После 2^{32} отсчёт порядковых номеров возобновляется с 0. Само значение sequence_number не включается в пакет, передаваемый через коммуникационную среду.

Алгоритмы MAC для каждого направления **должны** работать независимо и реализации **должны** обеспечивать возможность независимого выбора алгоритма для каждого из направления. Однако на практике **рекомендуется** использовать один алгоритм для обоих направлений.

Значение mac, возвращённое алгоритмом MAC, **должно** передаваться без шифрования, как заключительная часть пакета. Число байтов в поле mac зависит от выбранного алгоритма.

Определённые к настоящему времени алгоритмы MAC приведены в таблице.

Алгоритм	*Требование*	*Параметры*
hmac-sha1	**Обязательно**	HMAC-SHA1 (digest length = key length = 20)
hmac-sha1-96	**Рекомендуется**	Первые 96 битов HMAC-SHA1 (digest length = 12, key length = 20)
hmac-md5	**Не обязательно**	HMAC-MD5 (digest length = key length = 16)
hmac-md5-96	**Не обязательно**	HMAC-MD5 HMAC-MD5 (digest length = key length = 16)
none	**Не обязательно**	Без MAC, **не рекомендуется**

Алгоритмы "hmac-*" описаны в документе [RFC2104]. Алгоритмы "*-n" используют только первые n битов полученного значения.

Алгоритм SHA-1 описан в [FIPS-180-2], а MD5 - в [RFC1321].

Могут быть определены дополнительные методы защиты целостности в соответствии с [SSH-ARCH] и [SSH-NUMBERS].

6.5. Методы обмена ключами

Метод обмена ключей задаёт способ генерации одноразовых сеансовых ключей для шифрования и аутентификации, а также способ аутентификации.

Определены два **обязательных** метода обмена ключами.

`diffie-hellman-group1-sha1`	**Обязательно**
`diffie-hellman-group14-sha1`	**Обязательно**

Оба метода описаны в разделе 8.

Могут быть определены дополнительные методы, как описано в [SSH-NUMBERS]. Имя "diffie-hellman-group1-sha1" служит для обозначения метода обмена ключами, принятого группой Oakley, как определено в [RFC2409]. SSH поддерживает своё пространство имён идентификаторов групп, которое логически отличается от Oakley [RFC2412] и IKE, однако для одной дополнительной группы был адаптировано номер, выделенный в [RFC3526], а имя diffie-hellman-group14-sha1 используется для второй определённой группы. Реализациям следует трактовать эти номера, как неинтерпретируемые идентификаторы и не следует предполагать каких-либо связей между группами SSH и IKE.

6.6. Алгоритмы открытых ключей

Этот протокол был разработан для работы практически со всеми форматами, методами кодирования и алгоритмами (шифрование и подписи) открытых ключей.

Тип открытого ключа определяют несколько аспектов:

- Формат ключа - способ кодирования ключа и представления сертификатов; блоки данных (blob) ключей **могут** в дополнение к ключам содержать сертификаты;
- Алгоритм шифрования и/или подписи - некоторые типы алгоритмов могут не поддерживать одновременно шифрование и цифровые подписи; использование ключей может также ограничиваться требованиями политики (например, в сертификатах); в этом случае **следует** определять разные ключи для различных вариантов политики.
- Кодирование подписей и/или зашифрованных данных, включая (но не ограничиваясь) заполнение, порядок байтов и форматы данных.

К настоящему времени определены несколько форматов открытых ключей и/или сертификатов, показанных в таблице.

`ssh-dss`	`Обязательно`	`подпись`	`Ключ Raw DSS`
`ssh-rsa`	`Рекомендуется`	`подпись`	`Ключ Raw RSA`
`pgp-sign-rsa`	`Не обязательно`	`подпись`	`Сертификаты OpenPGP (ключ RSA)`
`pgp-sign-dss`	`Не обязательно`	`подпись`	`Сертификаты OpenPGP (ключ DSS)`

Могут быть определены дополнительные типы ключей в соответствии с [SSH-ARCH] и [SSH-NUMBERS].

Тип ключа **должен** быть всегда известен явно (в результате согласования алгоритма или иным способом). Обычно он включается в blob ключа.

Кодирование сертификатов и открытых ключей показано ниже.

```
string    идентификатор формата сертификата или открытого ключа
byte[n]   данные сертификата/ключа
```

Сертификатная часть может быть строкой нулевого размера, но открытый ключ является обязательным. Этот открытый ключ будет использоваться для аутентификации. Сертификат, содержащихся в блоке данных (blob) может использоваться для проверки полномочий.

Форматы сертификатов/открытых ключей, которые явно не задают формат подписи, **должны** использовать идентификатор открытого ключа/сертификата в качестве идентификатора цифровой подписи.

Цифровые подписи представляются в формате

```
string    идентификатор формата подписи (как задано в формате открытого ключа/сертификата)
byte[n]   blob цифровой подписи в соответствующем формату представлении
```

Формат ключа "ssh-dss" имеет показанное ниже представление

```
string    "ssh-dss"
mpint     p
mpint     q
mpint     g
mpint     y
```

где p, q, g и y - параметры из блока данных (blob) ключа подписи.

Подписывание и проверка с использованием этого формата ключа выполняются в соответствии со стандартом цифровой подписи DSS[1] [FIPS-186-2] с хэшем SHA-1 [FIPS-180-2].

Полученная подпись представляется следующим образом:

```
string    "ssh-dss"
string    dss_signature_blob
```

Значение dss_signature_blob представляется в форме строки, содержащей значение r, за которым следует s (160-битовое целое число без полей размера и заполнения с сетевым порядком байтов).

Формат ключа "ssh-rsa" использует представление

```
string    "ssh-rsa"
mpint     e
mpint     n
```

Где e и n - параметры из блока данных (blob) ключа подписи.

Подписывание и проверка с использованием этого формата ключа выполняются в соответствии со схемой RSASSA-PKCS1-v1_5 из [RFC3447] с хэшем SHA-1.

Полученная подпись представляется следующим образом:

```
string    "ssh-rsa"
string    rsa_signature_blob
```

Значение rsa_signature_blob представляется в форме строки, содержащей s (целое число без полей размера и заполнения с сетевым порядком байтов).

Метод "pgp-sign-rsa" указывает на сертификат, открытый ключ и подпись в двоичном формате, совместимом с OpenPGP ([RFC2440]). Этот метод указывает, что используется ключ RSA.

Метод "pgp-sign-dss" аналогичен предыдущему, но указывает на использование ключа DSS.

7. Обмен ключами

Обмен ключами (kex[2]) начинается каждой стороной с передачи списка имён (name-list) поддерживаемых алгоритмов. Каждая из сторон имеет предпочтительный алгоритм каждой категории и предполагается, что большинство реализаций в любой заданный момент будут использовать один и тот же предпочтительный алгоритм. Каждая сторона **может** предположить, какой алгоритм использует другая сторона и передать в соответствии с этим начальный пакет обмена, если данный алгоритм подходит в качестве предпочтительного метода.

Предположение считается ошибочным, если выполняется любое из условий:

- алгоритм kex и/или алгоритм ключа хоста предсказан неверно (предпочтительные алгоритмы клиента и сервера не совпали);
- если любые другие алгоритмы не могут быть согласованы (процедура определена ниже в параграфе 7.1).

В остальных случаях предположение считается верным и переданный пакет с предсказанием **должен** обрабатываться, как первый пакет обмена ключами.

Однако, если предсказание оказалось ошибочным и пакеты с предсказанием были переданы одной или обеими сторонами, такие пакеты **должны** игнорироваться (даже в тех случаях, когда ошибка в предсказании не воздействует на содержимое изначальных пакетов) и соответствующая сторона **должна** передать корректный исходный пакет.

Метод обмена ключами использует явную аутентификацию сервера, если сообщения при обмене включают сигнатуру или иные средства подтверждения аутентичности сервера. При обмене ключами используется неявная аутентификация сервера, если для подтверждения своей аутентичности сервер также предоставляет используемый разделяемый секрет K, передавая сообщение и соответствующий код MAC, которые клиент может проверить.

Метод обмена ключами, определённый в этом документе использует явную аутентификацию сервера. Однако методы с неявной аутентификацией также **могут** использоваться данным протоколом. После обмена ключами с неявной аутентификацией сервера клиент **должен** дождаться ответа сервера на свой запрос службы прежде, чем передавать какие-либо иные данные.

7.1. Согласование алгоритма

Обмен ключами с каждой стороны начинается передачей пакета, показанного на рисунке.

[1] Digital Signature Standard.
[2] Key exchange.

```
byte         SSH_MSG_KEXINIT
byte[16]     cookie (случайные байты)
name-list    kex_algorithms
name-list    server_host_key_algorithms
name-list    encryption_algorithms_client_to_server
name-list    encryption_algorithms_server_to_client
name-list    mac_algorithms_client_to_server
name-list    mac_algorithms_server_to_client
name-list    compression_algorithms_client_to_server
name-list    compression_algorithms_server_to_client
name-list    languages_client_to_server
name-list    languages_server_to_client
boolean      first_kex_packet_follows
uint32       0 (резерв для будущего расширения)
```

Каждое из полей name-list должно содержать список разделённых запятыми имён алгоритмов (см. Именование алгоритмов в [SSH-ARCH] и дополнительную информацию в [SSH-NUMBERS]). Все поддерживаемые (разрешённые) алгоритмы **должны** перечисляться в порядке снижения предпочтительности.

Первым в каждом поле **должен** указываться предпочтительный (предсказываемый) алгоритм. Каждое поле name-list **должно** содержать по крайней мере одно имя алгоритма.

cookie

Поле cookie **должно** содержать случайное значение, генерируемое сервером. Это поле служит для того, чтобы ни одна из сторон не могла полностью определить ключи и идентификатор сессии.

kex_algorithms

Алгоритмы обмена ключами, определённые выше. Первым должен указываться предпочитаемый (или предсказываемый) алгоритм. Если обе стороны использовали одинаковое «предсказание», **должен** использоваться именно этот алгоритм. В остальных случаях **должен** выполняться перебор указанных клиентом алгоритмов kex, пока не будет выполнено одно из приведённых ниже условий:

+ сервер также поддерживает этот алгоритм;
+ алгоритм требует пригодного для шифрования ключа хоста, такой алгоритм имеется в серверном списке server_host_key_algorithms и поддерживается клиентом;
+ алгоритм требует пригодного для подписи ключа хоста, такой алгоритм имеется в серверном списке server_host_key_algorithms и поддерживается клиентом.

Если подходящего алгоритма найти не удалось, соединение считается неудачным и обе стороны **должны** его разорвать.

server_host_key_algorithms

Список алгоритмов, поддерживаемых для серверного ключа хоста. Сервер перечисляет все алгоритмы, которые у него есть для ключа хоста. Хост может иметь множество ключей с одним или разными алгоритмами.

Некоторые из ключей хостов могут не поддерживать одновременно подписи и шифрование (это можно определить по алгоритму) и, таким образом, не все ключи хоста подходят для всех методов обмена ключами.

Выбор алгоритма зависит от того, требует ли выбранный алгоритм обмена ключами поддержки подписей и шифрования ключом хоста. Такое определение **должно** быть возможно по публичному имени алгоритма ключа. Выбираться **должен** первый алгоритм из клиентского списка, удовлетворяющий требованиям и поддерживаемый сервером. Если такого алгоритма не найдено, обе стороны **должны** отсоединиться.

encryption_algorithms

Список подходящих симметричных алгоритмов шифрования (шифров - cipher) в порядке снижения предпочтительности. Выбираемый для каждого направления алгоритм **должен** быть первым алгоритмом из клиентского списка, который поддерживается и сервером. Если такого алгоритма не найдено, обе стороны **должны** отсоединиться.

Отметим, что алгоритм "none" должен указываться явно, если его применение допустимо. Имена определённых алгоритмов перечислены в параграфе 6.3.

mac_algorithms

Список приемлемых алгоритмов MAC в порядке снижения предпочтительности. Выбираться **должен** первый алгоритм из клиентского списка, который присутствует и в серверном списке. Если такого алгоритма не найдено, обе стороны **должны** отсоединиться.

Отметим, что алгоритм "none" должен указываться явно, если его применение допустимо. Имена алгоритмов MAC перечислены в параграфе 6.4.

compression_algorithms

Список приемлемых алгоритмов компрессии в порядке снижения предпочтительности. Выбираться **должен** первый алгоритм из клиентского списка, который присутствует и в серверном списке. Если такого алгоритма не найдено, обе стороны **должны** отсоединиться.

Отметим, что алгоритм "none" должен указываться явно, если его применение допустимо. Имена алгоритмов компрессии перечислены в параграфе 6.2.

languages

Список приемлемых тегов языка [RFC3066] в порядке снижения предпочтительности. Обе стороны **могут** игнорировать этот список. Если предпочтительного языка нет, этот параметр name-list **следует** оставить пустым, как определено в разделе 5 [SSH-ARCH]. Теги языка **не следует** использовать, пока у передающей стороны нет уверенности в том, что они нужны.

first_kex_packet_follows

Указывает, будет ли далее передаваться предсказанный пакет обмена ключами. Если будет передан предсказанный пакет, поле **должно** иметь значение TRUE, в противном случае - FALSE.

После получения от другой стороны пакета SSH_MSG_KEXINIT каждая из сторон будет знать о корректности своего предсказания. Если предсказание другой стороны оказалось неверным, а данное поле имеет значение TRUE, следующий пакет **должен** отбрасываться без уведомления и обе стороны **должны** действовать в соответствии с требованиями согласованного метода обмена ключами. Если предсказание было верным, обмен ключами **должен** продолжаться с использованием предсказанного пакета.

После обмена сообщениями SSH_MSG_KEXINIT работает механизм обмена ключами, включающий обмен несколькими пакетами в соответствии со спецификацией данного алгоритма.

После того, как сторона передала сообщение SSH_MSG_KEXINIT для обмена (возможно, повторного) ключами и до момента передачи этой стороной сообщения SSH_MSG_NEWKEYS (параграф 7.3), для неё **недопустимо** передавать какие-либо сообщения, кроме:

- базовых сообщений транспортного уровня (1 - 19) (сообщения SSH_MSG_SERVICE_REQUEST и SSH_MSG_SERVICE_ACCEPT передавать **недопустимо**);
- сообщений согласования алгоритма (20 - 29) (но дополнительные сообщения SSH_MSG_KEXINIT передавать **недопустимо**);
- специфические для метода обмена ключами сообщения (30 - 49).

Для нераспознанных сообщений применяются рекомендации раздела 11.

Отметим, что при повторном обмене ключами после передачи сообщения SSH_MSG_KEXINIT каждая из сторон **должна** быть готова к обработке произвольного числа сообщений, которые могли оставаться в сети перед отправкой сообщения SSH_MSG_KEXINIT другой стороной.

7.2. Вывод при обмене ключами

В результате обмена ключами создаются два значения: разделяемый секрет K и хэш обмена H. Из них получаются ключи шифрования и аутентификации. Хэш обмена H из первого обмена ключами используется также в качестве идентификатора сессии, уникально обозначающего данное соединение. Этот идентификатор используется методами аутентификации, как часть данных, которые подписываются в качестве доказательства владения секретным ключом. После расчёта идентификатора сессии его значение не меняется даже при повторе обмена ключами.

Каждый метод обмена ключами задаёт хэш-функцию, используемую при обмене. Тот же алгоритм хэширования **должен** использоваться при создании ключа. Далее эта функция обозначается HASH.

Ключи шифрования **должны** рассчитываться, как HASH известного значения K следующим образом:

- Начальный вектор IV от клиента к серверу: HASH(K || H || "A" || session_id) (K представляется, как mpint, "A", как байт, и session_id, как «неразобранные» (raw) данные. "A" означает одиночный символ A, ASCII 65).
- Начальный вектор IV от сервера к клиенту: HASH(K || H || "B" || session_id)
- Ключ шифрования от клиента к серверу: HASH(K || H || "C" || session_id)
- Ключ шифрования от сервера к клиенту: HASH(K || H || "D" || session_id)
- Ключ контроля целостности от клиента к серверу: HASH(K || H || "E" || session_id)
- Ключ контроля целостности от сервера к клиенту: HASH(K || H || "F" || session_id)

Данные ключа **должны** браться из начала вывода хэш-функции в требуемом количестве. Если размер ключа превышает размер результата HASH, ключ расширяется путём расчёта HASH для конкатенации K, H и полученных ранее хэш-значений с добавлением результата с конец имеющегося ключа. Этот процесс повторяется, пока не будет получен достаточный объем материала, из начальной части которого и берётся ключ. Иными словами:

```
K1 = HASH(K || H || X || session_id)   (X равно, например, "A")
K2 = HASH(K || H || K1)
K3 = HASH(K || H || K1 || K2)
...
key = K1 || K2 || K3 || ...
```

Это процесс приведёт к потере энтропии, если объем энтропии K больше, чем размер внутреннего состояния HASH.

7.3. Принятие ключей для использования

Обмен ключами каждая сторона завершает передачей сообщения SSH_MSG_NEWKEYS. Это сообщение передаётся со старыми ключами и алгоритмами. Все сообщения, передаваемые после него, **должны** использовать новые ключи и алгоритмы.

Когда это сообщение получено, для приёма **должны** использоваться новые ключи и алгоритмы.

Это сообщение предназначено для обеспечения возможности ответить сообщением SSH_MSG_DISCONNECT, которое другая сторона сможет понять, если при обмене ключами возникнут какие-либо проблемы.

```
byte      SSH_MSG_NEWKEYS
```

8. Обмен ключами по методу Diffie-Hellman

Обмен ключами по методу Diffie-Hellman (DH) обеспечивает разделяемый секрет, который не может быть определён третьей стороной. Обмен ключами комбинируется с сигнатурой, использующей ключ хоста для аутентификации хоста. Этот метод обмена ключами обеспечивает явную аутентификацию сервера, как определено в разделе 7.

Этапы обмена ключами перечислены ниже. Пусть C - клиент, S - сервер, p - большое безопасное простое число, g - генератор для подгруппы GF(p), q - порядок подгруппы, V_S - строка идентификации сервера S, V_C - строка идентификации клиента C, K_S - открытый ключ сервера S, I_C сообщение SSH_MSG_KEXINIT клиента C и I_S - сообщение SSH_MSG_KEXINIT сервера S, обмен которыми произошёл до начала описываемого ниже.

1. C генерирует случайное значение x (1 < x < q) и рассчитывает e = g^x mod p, а потом передаёт e серверу S.
2. S генерирует случайное значение y (0 < y < q) и рассчитывает f = g^y mod p. S получает e, рассчитывает K = e^y mod p, H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K) (эти элементы кодируются в соответствии с их типами, как показано ниже) и подпись s на H с секретным ключом хоста. S передаёт (K_S || f || s) клиенту C. Операция подписания может включать вторую операцию хэширования.

3. C убеждается, что K_S реально является ключом хоста S (например, используя сертификаты или локальную базу данных). C также может воспринять ключ без верификации, однако это создаётся уязвимость для активных атак (на практике, тем не менее, это может кратковременно использоваться во многих средах). После этого C рассчитывает K = f^x mod p, H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K) и проверяет подпись s на H.

Значения e или f, выходящие из диапазона [1, p-1], **недопустимо** передавать или воспринимать на другой стороне. Если это условие нарушается, обмен ключами приводит к отказу.

Реализация этого выполняется с помощью показанных ниже сообщений. Алгоритм хэширования для расчёта хэш-функции обмена определяется именем метода и далее обозначается HASH. Алгоритм с открытым ключом для подписи согласуется с помощью сообщений SSH_MSG_KEXINIT.

Сначала клиент передаёт сообщение

```
byte      SSH_MSG_KEXDH_INIT
mpint     e
```

На которое сервер отвечает сообщением

```
byte      SSH_MSG_KEXDH_REPLY
string    открытый ключ сервера и сертификаты K_S)
mpint     f
string    подпись H
```

Хэш H рассчитывается, как HASH для конкатенации приведённых ниже переменных:

```
string    V_C, строка идентификации клиента (CR и LF исключены)
string    V_S, строка идентификации сервера (CR и LF исключены)
string    I_C, данные клиентского SSH_MSG_KEXINIT
string    I_S, данные серверного SSH_MSG_KEXINIT
string    K_S, ключ хоста
mpint     e, обменное значение, передаваемое клиентом
mpint     f, обменное значение, передаваемое сервером
mpint     K, разделяемый секрет
```

Это значение называется хэшем обмена (exchange hash) и используется для аутентификации обмена ключами. Хэш обмена **следует** держать в секрете.

Алгоритм цифровой подписи **должен** применяться к H, а не исходным данным. Большинство алгоритмов цифровой подписи включает хэширование и дополнительное заполнение (например, "ssh-dss" задаёт хэширование SHA-1). В таком случае данные сначала хэшируются с помощью HASH для расчёта H, а потом значение H хэшируется с помощью SHA-1 в процессе создания подписи.

8.1. diffie-hellman-group1-sha1

Метод "diffie-hellman-group1-sha1" задаёт обмен ключами Diffie-Hellman с использованием алгоритма SHA-1 в качестве HASH и Oakley Group 2 [RFC2409] (1024-битовая группа MODP). Этот метод **должен** поддерживаться для взаимодействия, поскольку все известные реализации поддерживают его. Отметим, что в обозначении метода указано "group1", хотя он задаёт использование Oakley Group 2.

8.2. diffie-hellman-group14-sha1

Метод "diffie-hellman-group14-sha1" задаёт обмен ключами Diffie-Hellman с использованием алгоритма SHA-1 в качестве HASH и Oakley Group 14 [RFC3526] (2048-битовая группа MODP). Метод также **должен** поддерживаться.

9. Повторный обмен ключами

Повторный обмен ключами начинается с передачи пакета SSH_MSG_KEXINIT, когда уже не происходит обмена ключами (как описано в параграфе 7.1). При получении такого сообщения приёмная сторона **должна** ответить своим сообщением SSH_MSG_KEXINIT, если полученное сообщение SSH_MSG_KEXINIT само не было таким ответом. Повторный обмен ключами **может** инициировать любая из сторон, но замена их ролей **недопустима** (т. е., сервер остаётся сервером, а клиент - клиентом).

Повторный обмен ключами выполняется с использованием шифрования, которое действовало на момент начала этого обмена. Методы шифрования, компрессии и MAC не меняются до передачи нового сообщения SSH_MSG_NEWKEYS после обмена ключами (как при начальном обмене). Повторный обмен выполняется аналогично первоначальному за исключением того, что идентификатор сессии не изменяется. Можно сменить некоторые или все алгоритмы в процессе повторного обмена ключами. Ключи хостов также можно менять. Все ключи и векторы инициализации заново рассчитываются после обмена. Контексты алгоритмов компрессии и шифрования сбрасываются.

Рекомендуется менять ключи после каждого гигабайта переданной информации или после каждого часа работы соединения (что произойдёт раньше). Однако, поскольку повторный обмен осуществляется с использованием открытых ключей, он требует значительных вычислительных ресурсов и не следует менять ключи слишком часто.

После передачи пакета SSH_MSG_NEWKEYS можно продолжать передачу данных от приложений - смена ключей не оказывает влияния на протоколы, работающие «поверх» транспортного уровня SSH.

10. Запрос обслуживания

После обмена ключами клиент запрашивает обслуживание. Службы идентифицируются по именам. Формат имён и процедуры определения новых имён описаны в [SSH-ARCH] и [SSH-NUMBERS].

В настоящее время зарезервированы два имени служб:

```
ssh-userauth
ssh-connection
```

Для именования локальных служб применяется политика, подобная политике локального именования алгоритмов. Для локальных служб следует применять синтаксис выделенных для **приватного использования** (PRIVATE USE) имён "servicename@domain".

```
byte      SSH_MSG_SERVICE_REQUEST
string    имя службы
```

Если сервер отвергает запрос на обслуживание, ему **следует** передать подходящее сообщение SSH_MSG_DISCONNECT, после чего сервер **должен** отсоединиться.

При старте службы она может иметь доступ к идентификатору сессии, созданному при обмене ключами.

Если сервер поддерживает службу (и позволяет клиенту использовать её), он **должен** ответить сообщением:

```
byte      SSH_MSG_SERVICE_ACCEPT
string    имя службы
```

Номера сообщений, используемых службами, следует брать из областей, зарезервированных для этого (см. [SSH-ARCH] и [SSH-NUMBERS]). Транспортный уровень будет продолжать обработку своих сообщений.

Отметим, что после обмена ключами с неявной аутентификацией сервера клиент **должен** дождаться отклика на свой запрос службы прежде, чем передавать какие-либо данные.

11. Дополнительные сообщения

Каждая из сторон может передавать любые дополнительные сообщения в любой момент.

11.1. Сообщение о разъединении

```
byte      SSH_MSG_DISCONNECT
uint32    код причины
string    описание в кодировке ISO-10646 UTF-8 [RFC3629]
string    тег языка [RFC3066]
```

Это сообщение вызывает незамедлительный разрыв соединения. Все реализации **должны** быть способны обрабатывать такие сообщения, **следует** также обеспечивать возможность передачи таких сообщений.

Отправителю сообщения **недопустимо** передавать или принимать какие-либо данные после передачи этого сообщения, а получателю **недопустимо** воспринимать какие-либо данные после получения такого сообщения. Поле описания (description) содержит информацию о причине разрыва в понятной человеку форме. Поле reason code содержит код причины в машинном формате (пригоден для локализации). Возможные значение кодов приведены в таблице (для удобства коды в таблице приведены в десятичном формате, хотя на деле применяются шестнадцатеричные коды в формате uint32).

Имя сообщения	*Код причины*
SSH_DISCONNECT_HOST_NOT_ALLOWED_TO_CONNECT	1
SSH_DISCONNECT_PROTOCOL_ERROR	2
SSH_DISCONNECT_KEY_EXCHANGE_FAILED	3
SSH_DISCONNECT_RESERVED	4
SSH_DISCONNECT_MAC_ERROR	5
SSH_DISCONNECT_COMPRESSION_ERROR	6
SSH_DISCONNECT_SERVICE_NOT_AVAILABLE	7
SSH_DISCONNECT_PROTOCOL_VERSION_NOT_SUPPORTED	8
SSH_DISCONNECT_HOST_KEY_NOT_VERIFIABLE	9
SSH_DISCONNECT_CONNECTION_LOST	10
SSH_DISCONNECT_BY_APPLICATION	11
SSH_DISCONNECT_TOO_MANY_CONNECTIONS	12
SSH_DISCONNECT_AUTH_CANCELLED_BY_USER	13
SSH_DISCONNECT_NO_MORE_AUTH_METHODS_AVAILABLE	14
SSH_DISCONNECT_ILLEGAL_USER_NAME	15

При отображении строки description следует использовать фильтрацию управляющих символов, описанную в [SSH-ARCH], для предотвращения атак с использованием символов терминального управления.

Запросы на выделение новых значений кодов причины разрыва соединения (reason code) из диапазона 0x00000010 - 0xFDFFFFFF и связанных с ними описаний (description) **должны** выполняться по процедуре IETF CONSENSUS (согласование с IETF), как описано в [RFC2434]. Значения кодов причины из диапазона 0xFE000000 - 0xFFFFFFFF предназначены для **приватного использования** (PRIVATE USE). Как было отмечено выше, актуальные инструкции для IANA по распределению новых значений приведены в [SSH-NUMBERS].

11.2. Сообщение о проигнорированных данных

```
byte      SSH_MSG_IGNORE
string    данные
```

Все реализации **должны** понимать (и игнорировать) такие сообщения в любой момент (после получения строки идентификации). Передача таких сообщений от реализаций не требуется. Эти сообщения могут служить дополнительной защитой от средств анализа трафика.

11.3. Отладочное сообщение

```
byte      SSH_MSG_DEBUG
boolean   always_display
string    сообщение в кодировке ISO-10646 UTF-8 [RFC3629]
string    тег языка [RFC3066]
```

Все реализации **должны** понимать эти сообщения, но могут игнорировать их. Сообщения служат для передачи информации, способной помочь в отладке. Если поле always_display имеет значение TRUE, сообщение **следует**

вывести на экран. В противном случае отображать сообщения **не следует**, если отладочная информация явно не запрошена пользователем.

Поле сообщения (message) не обязано содержать новую строку. Однако оно может включать множество строк, разделённых парами символов CRLF (возврат каретки - перевод строки).

При отображении строки message следует использовать фильтрацию управляющих символов, описанную в [SSH-ARCH], для предотвращения атак с использованием символов терминального управления.

11.4. Резервные сообщения

Реализации **должны** отвечать на все нераспознанные сообщения своими сообщениями SSH_MSG_UNIMPLEMENTED, порядок которых должен соответствовать порядку непонятых сообщений. В противном случае такие сообщения **должны** игнорироваться. Последующие версии протокола могут определить иную трактовку сообщений этого типа.

```
byte      SSH_MSG_UNIMPLEMENTED
uint32    порядковый номер отвергнутого пакета
```

12. Номера сообщений

В таблице справа приведён список сообщений и связанных с ними номеров.

SSH_MSG_DISCONNECT	1
SSH_MSG_IGNORE	2
SSH_MSG_UNIMPLEMENTED	3
SSH_MSG_DEBUG	4
SSH_MSG_SERVICE_REQUEST	5
SSH_MSG_SERVICE_ACCEPT	6
SSH_MSG_KEXINIT	20
SSH_MSG_NEWKEYS	21

Отметим, что номера 30-49 используются для пакетов kex. Разные методы согласования ключей могут использовать совпадающие номера из этого диапазона.

13. Согласование с IANA

Этот документ является частью согласованного набора документов. Вопросы согласования с агентством IANA для протокола SSH, определённые в [SSH-ARCH], [SSH-USERAUTH], [SSH-CONNECT] и настоящем документе, детализированы в [SSH-NUMBERS].

14. Вопросы безопасности

Протокол обеспечивает защищённые шифрованные каналы через сеть без защиты. Он обеспечивает аутентификацию серверных хостов, обмен ключами, шифрование и защиту целостности данных. Протокол также обеспечивает уникальные идентификаторы сессий, которые могут использоваться протоколами вышележащих уровней.

Полное рассмотрение вопросов безопасности для этого протокола приведено в [SSH-ARCH].

15. Литература

15.1. Нормативные документы

[SSH-ARCH] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[SSH-USERAUTH] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[SSH-CONNECT] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.

[SSH-NUMBERS] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm ", RFC 1321, April 1992.

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2144] Adams, C., "The CAST-128 Encryption Algorithm", RFC 2144, May 1997.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2440] Callas, J., Donnerhacke, L., Finney, H., and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[FIPS-180-2] US National Institute of Standards and Technology, "Secure Hash Standard (SHS)", Federal Information Processing Standards Publication 180-2, August 2002.

[FIPS-186-2] US National Institute of Standards and Technology, "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-2, January 2000.

[FIPS-197] US National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication 197, November 2001.

[FIPS-46-3] US National Institute of Standards and Technology, "Data Encryption Standard (DES)", Federal Information Processing Standards Publication 46-3, October 1999.

[SCHNEIER] Schneier, B., "Applied Cryptography Second Edition: protocols algorithms and source in code in C", John Wiley and Sons, New York, NY, 1996.

[TWOFISH] Schneier, B., "The Twofish Encryptions Algorithm: A 128-Bit Block Cipher, 1st Edition", March 1999.

15.2. Дополнительная литература

[RFC0894] Hornig, C., "Standard for the transmission of IP datagrams over Ethernet networks", STD 41, RFC 894, April 1984.

[RFC1661] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[RFC2412] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[ssh-1.2.30] Ylonen, T., "ssh-1.2.30/RFC", Файл в архиве ftp://ftp.funet.fi/pub/unix/security/login/ssh/ssh-1.2.30.tar.gz, November 1995.

Адреса авторов

Tatu Ylonen
SSH Communications Security Corp
Valimotie 17
00380 Helsinki
Finland
EMail: ylo@ssh.com

Chris Lonvick (редактор)
Cisco Systems, Inc.
12515 Research Blvd.
Austin 78759
USA
EMail: clonvick@cisco.com

Перевод на русский язык

Николай Малых

nmalykh@gmail.com

Торговые марки

ssh – торговый знак, зарегистрированный в США и/или других странах.

Финансирование функций RFC Editor обеспечено IETF Administrative Support Activity (IASA).