ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

РЕКОМЕНДАЦИИ ПО СТАНДАРТИЗАЦИИ

Р 1323565.1.030— 2020

Информационная технология

КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ

Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.3)

Издание официальное

Москва
Стандартинформ
2020

gost.e-ecolog.ru

Предисловие

1 РАЗРАБОТАНЫ Обществом с ограниченной ответственностью «КРИПТО-ПРО» (ООО «КРИПТО-ПРО»)

2 ВНЕСЕНЫ Техническим комитетом по стандартизации ТК 26 «Криптографическая защита информации»

3 УТВЕРЖДЕНЫ И ВВЕДЕНЫ В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 27 февраля 2020 г. № 84-ст

4 ВВЕДЕНЫ ВПЕРВЫЕ

*Правила применения настоящих рекомендаций установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящим рекомендациям публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящих рекомендаций соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (*gost.e-ecolog.ru

© ISO, 2014 — Все права сохраняются
© Стандартинформ, оформление, 2020

Настоящие рекомендации не могут быть полностью или частично воспроизведены, тиражированы и распространены в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

1 Область применения . . . 1
2 Нормативные ссылки . . . 1
3 Термины, определения, обозначения и сокращения . . . 2
3.1 Термины и определения . . . 2
3.2 Обозначения . . . 3
3.3 Сокращения . . . 5
4 Обзор протокола TLS . . . 5
4.1 Иерархия информационного обмена . . . 5
4.2 Состояние соединения . . . 6
5 Протокол Handshake . . . 6
5.1 Формат сообщений протокола Handshake . . . 6
5.2 Базовые принципы работы протокола Handshake . . . 8
5.3 Схемы аутентифицированной выработки общего ключевого материала . . . 10
5.4 Пересогласование открытых эфемерных ключей . . . 15
5.5 Сообщения ключевого обмена . . . 16
5.6 Расширения . . . 20
5.7 Параметры сервера . . . 31
5.8 Сообщения аутентификации . . . 32
5.9 Post-handshake сообщения . . . 35
6 Протокол Record . . . 39
6.1 Фрагментация . . . 40
6.2 Формирование записи . . . 40
6.3 Защита данных . . . 42
6.4 Счетчик полученных/отправленных записей . . . 43
6.5 Дополнение данных . . . 44
7 Протокол Alert . . . 44
7.1 Оповещения закрытия соединения . . . 45
7.2 Оповещения об ошибках . . . 46
8 Криптографические вычисления . . . 48
8.1 Функции, используемые при выработке ключей . . . 48
8.2 Иерархия ключей . . . 49
8.3 Обновление секретных значений . . . 52
8.4 Ключевой материал трафика . . . 53
8.5 Выработка общего секретного значения *ECDHE* . . . 53
8.6 Выработка предварительно распределенного секрета *PSK* . . . 54
8.7 Экспорт ключевого материала . . . 55
8.8 Функция *Transcript-Hash* . . . 55
8.9 Значения *Handshake Context* и *Finished Secret* . . . 55
9 Прикладные данные . . . 56
10 Использование российских криптографических алгоритмов . . . 56
10.1 Идентификаторы криптонаборов из реестра «TLSCipherSuites» . . . 56
10.2 Идентификаторы схем подписи из реестра «TLSSignatureScheme» . . . 59
10.3 Идентификаторы кривых из реестра «TLSSupportedGroups» . . . 60

11 Вопросы реализации и безопасности 61
11.1 Механизмы защиты от атак по побочным каналам 61
11.2 Механизмы защиты от downgrade-атак 61
Приложение А (справочное) Рекомендации по использованию TLS 1.3 криптонаборов в СКЗИ 63
Приложение Б (справочное) Язык представления данных в протоколе TLS 64

Введение

Настоящие рекомендации содержат описание протокола безопасности транспортного уровня версии 1.3 (TLS 1.3) с криптонаборами на основе алгоритмов блочного шифрования «Магма» и «Кузнечик», описанных в ГОСТ Р 34.12.

Необходимость разработки настоящего документа вызвана потребностью в обеспечении совместимости различных реализаций протокола TLS 1.3 с использованием российских государственных криптографических стандартов.

Примечание — Основная часть настоящих рекомендаций дополнена приложениями А и Б.

Р Е К О М Е Н Д А Ц И И П О С Т А Н Д А Р Т И З А Ц И И

Информационная технология

КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ

Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.3)

Information technology. Cryptographic data security.
The use of the Russian cryptographic algorithms in the Transport Layer Security protocol (TLS 1.3)

Дата введения — 2020—06—01

1 Область применения

Настоящие рекомендации содержат описание протокола безопасности транспортного уровня (далее — TLS), соответствующего версии TLS 1.3, описанной в [1], и содержат описание соответствующих данному протоколу криптонаборов, предназначенных для установления защищенного соединения между клиент-серверными приложениями с использованием алгоритмов, определяемых российскими государственными криптографическими стандартами, для обеспечения аутентификации сторон, конфиденциальности и целостности информации, передаваемой по каналу связи.

2 Нормативные ссылки

В настоящих рекомендациях использованы нормативные ссылки на следующие стандарты:

ГОСТ Р 34.10—2012 Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи

ГОСТ Р 34.11—2012 Информационная технология. Криптографическая защита информации. Функция хэширования

ГОСТ Р 34.12 Информационная технология. Криптографическая защита информации. Блочные шифры

Р 50.1.113—2016 Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования

Р 1323565.1.012 Информационная технология. Криптографическая защита информации. Принципы разработки и модернизации шифровальных (криптографических) средств защиты информации

Р 1323565.1.020 Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.2)

Р 1323565.1.023 Информационная технология (ИТ). Криптографическая защита информации. Использование алгоритмов ГОСТ Р 34.10—2012, ГОСТ Р 34.11—2012 в сертификате, списке аннулированных сертификатов (CRL) и запросе на сертификат PKCS # 10 инфраструктуры открытых ключей X.509

Р 1323565.1.024 Информационная технология. Криптографическая защита информации. Параметры эллиптических кривых для криптографических алгоритмов и протоколов

Р 1323565.1.026 Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров, реализующие аутентифицированное шифрование

Издание официальное

Примечание — При пользовании настоящими рекомендациями целесообразно проверить действие ссылочных документов в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный документ, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого документа с учетом всех внесенных в данную версию изменений. Если заменен ссылочный документ, на который дана датированная ссылка, то рекомендуется использовать версию этого документа с указанным выше годом утверждения (принятия). Если после утверждения настоящих рекомендаций в ссылочный документ, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный документ отменен без замены, то положение, в котором дана ссылка на него, применяется в части, не затрагивающей эту ссылку.

3 Термины, определения, обозначения и сокращения

3.1 Термины и определения

В настоящих рекомендациях применены следующие термины с соответствующими определениями:

3.1.1 **клиент** (client): Сторона взаимодействия, инициирующая установление TLS-соединения.

3.1.2 **сервер** (server): Сторона взаимодействия, не инициирующая установление TLS-соединения.

3.1.3 **TLS-соединение** (connection): Соединение транспортного уровня между сторонами взаимодействия, возникающее в момент отправки клиентом сообщения приветствия ClientHello (далее — исходное сообщение ClientHello1 для данного соединения) и перестающее существовать при закрытии соединения с помощью пересылки оповещений (см. раздел 7) либо при внештатном обрыве связи.

Примечания

1 В настоящих рекомендациях установлено, что термины «соединение» и «TLS соединение» являются синонимами, если не оговорено иное.

2 При повторной отправке сообщения ClientHello (далее — повторное сообщение ClientHello2) в ответ на сообщение HelloRetryRequest (см. 5.5.3) новое соединение не создается.

3 В версии протокола TLS 1.3, описанной в текущих рекомендациях, термин «сессия», существовавший в предыдущих версиях протокола (см. Р 1323565.1.020), не используется, так как механизм возобновления соединения в настоящих рекомендациях заменен механизмом использования предварительно распределенного секрета.

3.1.4 **текущее соединение:** TLS-соединение, для которого исходное сообщение ClientHello1 было отправлено до текущего момента времени, но закрытия соединения еще не произошло.

3.1.5 **соединение, предшествующее текущему соединению:** TLScоединение, возникшее до момента пересылки исходного сообщения ClientHello1 для текущего соединения.

3.1.6 **инициализирующее соединение:** Соединение, предшествующее текущему соединению, в рамках которого было выработано предварительно согласованное секретное значение, используемое в рамках текущего соединения.

3.1.7 **трафик** (traffic): Поток данных, передающийся в соединении.

3.1.8 **криптонабор** (cipher suite): Набор криптографических алгоритмов и их параметров, определяющий работу протокола TLS в рамках соответствующего данному криптонабору соединения.

3.1.9 **согласованный криптонабор:** Криптонабор, соответствующий идентификатору криптонабора, согласованному сторонами взаимодействия в процессе выполнения протокола Handshake, описанного в разделе 5.

3.1.10 **TLS 1.3 сервер/клиент:** Сервер/клиент, поддерживающий протокол TLS 1.3.

3.1.11 **режим совместимости:** Режим работы протокола TLS 1.3, в рамках которого TLS 1.3 сервер/клиент допускает возможность работы с клиентом/сервером в рамках версии протокола TLS, предшествующей версии TLS 1.3.

3.1.12 **downgrade атака:** Атака, в результате которой клиент и сервер устанавливают соединения в рамках версии протокола, предшествующей максимальной версии, поддерживающейся клиентом и сервером.

3.1.13

ключ подписи: Элемент секретных данных, специфичный для субъекта и используемый только данным субъектом в процессе формирования цифровой подписи.

[ГОСТ Р 34.10—2012, статья 3.1.2]

3.1.14

ключ проверки подписи: Элемент данных, математически связанный с ключом подписи и используемый проверяющей стороной в процессе проверки цифровой подписи.

[ГОСТ Р 34.10—2012, статья 3.1.3]

3.1.15 **закрытый эфемерный ключ:** Элемент секретных данных, генерируемый случайным образом для каждого нового соединения и используемый только в рамках данного соединения. В рамках настоящих рекомендаций закрытый эфемерный ключ является числом, диапазон принимаемых значений которого задается в соответствии с выбранной эллиптической кривой (см. подробнее 8.5.1).

3.1.16 **открытый эфемерный ключ:** Элемент данных, математически связанный с эфемерным закрытым ключом, генерируемый для каждого нового соединения и используемый только в рамках данного соединения. В рамках настоящих рекомендаций открытый эфемерный ключ является парой координат (x, y), являющейся точкой некоторой эллиптической кривой (см. подробнее 8.5.1).

3.1.17 **прикладные данные** (application data): Данные прикладного уровня, пересылаемые в рамках работы протокола TLS (см. подробнее раздел 9).

3.1.18 **предварительно распределенный секрет:** энтропийные данные, используемые в некотором соединении для выработки общих секретных значений в соответствии с иерархией ключей (см. 8.2) и известные обеим сторонам до начала работы данного соединения.

Примечание — В настоящих рекомендациях установлено, что термины «предварительно распределенный секрет» и «PSK-значение» являются синонимами.

3.1.19 **внешний предварительно распределенный секрет:** предварительно распределенный секрет, выработанный сторонами вне протокола TLS.

3.1.20 **внутренний предварительно распределенный секрет:** предварительно распределенный секрет, выработанный сторонами в рамках протокола TLS.

Примечания

1 В настоящих рекомендациях в целях сохранения терминологической преемственности с действующими отечественными нормативными документами и опубликованными научно-техническими изданиями установлено, что термины «электронная подпись», «цифровая подпись», «электронная цифровая подпись» и «подпись» являются синонимами.

2 В настоящих рекомендациях используются обозначения параметров эллиптических кривых в соответствии с ГОСТ Р 34.10—2012 (раздел 5).

3 В настоящих рекомендациях под обозначениями «[*sender*]» и «[*receiver*]» подразумевается переменная, принимающая значения из множества {«*client*», «*server*»}. Например, под обозначением [*sender*]*_write_key* подразумевается переменная, принимающая значения из множества {*client_write_key*, *server_write_key*}.

3.2 Обозначения

В настоящих рекомендациях применены следующие обозначения:

B_s — множество байтовых строк длины s, $s \geq 0$. Строка $b = (b_1, \ldots, b_s)$ принадлежит множеству B_s, если $b_1, \ldots, b_s \in \{0, \ldots 255\}$. При $s = 0$ множество B_s состоит из единственной пустой строки длины 0;

B^* — множество всех байтовых строк произвольной конечной длины;

$|b|$ — длина байтовой строки $b \in B^*$ (если b — пустая строка, то $|b| = 0$);

$|$ — конкатенация двух байтовых строк; для двух строк $a = (a_1, \ldots, a_{s1}) \in B_{s1}$, $b = (b_1, \ldots, b_{s2}) \in B_{s2}$ их конкатенацией $a|b$ называется строка $c = (a_1, \ldots, a_{s1}, b_1, \ldots b_{s2}) \in B_{s1 + s2}$;

b^s — байтовая строка длины s вида $b^s = \underbrace{(b, b, \ldots, b)}_{s} \in B_s$, где $b \in B_1$;

$b[i..j]$ — строка $b[i..j] = (b_i, b_{i+1}, \ldots, b_j) \in B_{j-i+1}$, где $1 \le i \le j \le s$ и $b = (b_1, \ldots, b_s) \in B_s$;

$LMB_t(b)$ — строка $LMB_t(b) = (b_1, \ldots, b_t) \in B_t$, соответствующая строке $b = (b_1, \ldots, b_s) \in B_s$, $1 \le t \le s$;

$STR_s(r)$ — строка $STR_s(r) = (b_1, \ldots, b_s) \in B_s$, соответствующая числу $r = 256^{s-1} \cdot b_1 + \ldots + 256 \cdot b_{s-1} + b_s \le 256^s - 1$ (представление числа r в виде байтовой строки в формате big-endian);

$str_s(r)$ — строка $str_s(r) = (b_1, \ldots, b_s) \in B_s$, соответствующая числу $r = 256^{s-1} \cdot b_s + \ldots + 256 \cdot b_2 + b_1 \le 256^s - 1$ (представление числа r в виде байтовой строки в формате little-endian);

& — операция побитовой конъюнкции (побитовое "И");

$\lceil r \rceil$ — наименьшее целое число, большее или равное r;

n — параметр алгоритма блочного шифрования, называемый длиной блока, задаваемый согласованным криптонабором (см. 10.1.1); в рамках данного документа измеряется в байтах;

$KLen$ — параметр алгоритма блочного шифрования, называемый длиной ключа, задаваемый согласованным криптонабором (см. 10.1.1); в рамках данного документа измеряется в байтах;

$IVLen$ — длина вектора инициализации в байтах, задаваемая согласованным криптонабором (см. 10.1.2);

Q_c — открытый эфемерный ключ клиента;

d_c — закрытый эфемерный ключ клиента;

Q_s — открытый эфемерный ключ сервера;

d_s — закрытый эфемерный ключ сервера;

E_i — эллиптическая кривая, указанная клиентом в расширении supported_groups;

m_i — порядок группы точек эллиптической кривой E_i;

q_i — порядок циклической подгруппы группы точек эллиптической кривой E_i;

P_i — точка эллиптической кривой E_i порядка q_i;

h_i — кофактор циклической подгруппы порядка q_i группы точек эллиптической кривой E_i, значение которого равно $\frac{m_i}{q_i}$;

(d^i_s, Q^i_s) — эфемерная ключевая пара сервера: закрытый и открытый ключи, соответствующие кривой E_i;

(d^i_c, Q^i_c) — эфемерная ключевая пара клиента: закрытый и открытый ключи, соответствующие кривой E_i;

O_i — нулевая точка эллиптической кривой E_i;

Q_{verify} — ключ проверки подписи, хранящийся в сертификате стороны взаимодействия;

d_{sign} — ключ подписи, соответствующий ключу Q_{verify};

$SIGN$ — функция формирования подписи, принимающая на вход произвольную байтовую строку $M \in B^*$ и ключ подписи d_{sign} и выдающая в качестве результата своей работы строку sgn (см. 10.2);

$\oplus$ — операция покомпонентного сложения по модулю 2 двух байтовых строк одинаковой длины;

$HASH$ — хэш-функция, задаваемая согласованным криптонабором (см. 10.1.4);

$HLen$ — длина выхода хэш-функции $HASH$ в байтах (см. 10.1.4);

$HMAC$ — функция вычисления кода аутентификации сообщения, определяемая алгоритмом HMAC, описанным в Р 50.1.113—2016 (подраздел 4.1) и задающимся в соответствии с хэш-функцией, задаваемой согласованным криптонабором (см. 10.1.4).

Примечания

1 В настоящих рекомендациях все строковые константы приводятся в кавычках и представляются в кодировке ASCII без терминирующего нуль-символа.

2 В настоящих рекомендациях установлен следующий порядок перевода чисел в строки, если иное не оговаривается: все числовые значения, имеющие тип uint8, uint16, uint24, uint32, uint64, представляются в виде строк в

формате big-endian. Например, десятичное число 16909060 с типом uint32 представляется в виде байтовой строки 01 02 03 04 в шестнадцатеричном виде.

3 В настоящих рекомендациях установлено, что весь трафик, пересылаемый в канале связи, имеет байтовое представление.

4 Байтовое представление данных, соответствующих определенной структуре, задается конкатенацией байтовых представлений значений полей структуры в порядке их объявления (сверху вниз). Байтовое представление значения поля, являющегося вектором элементов некоторого типа, задается конкатенацией байтовых представлений элементов данного вектора в порядке их нумерации (слева направо). Все числовые значения представляются в виде байтовых строк в соответствии с примечанием 2.

5 В настоящих рекомендациях в целях сохранения терминологической преемственности с действующими отечественными нормативными документами и опубликованными научно-техническими изданиями установлено, что термины «хэш-функция», «криптографическая хэш-функция», «функция хэширования» и «криптографическая функция хэширования» являются синонимами.

6 Для описания протокола в настоящих рекомендациях используется общепринятый язык представления данных в протоколе TLS (далее язык представления TLS), описанный в приложении Б.

3.3 Сокращения

В настоящих рекомендациях применены следующие сокращения:

СКЗИ — средство криптографической защиты информации;

AEAD — (Authenticated Encryption with Associated Data) шифрование с имитозащитой и ассоциированными данными;

TCP — (Transmission Control Protocol) протокол управления передачей.

4 Обзор протокола TLS

Основное назначение протокола TLS — создание защищенного канала связи между двумя взаимодействующими сторонами, то есть обеспечение следующих свойств:

- аутентификация сторон: обеспечивается за счет проверки сформированного значения подписи или за счет подтверждения (неявного) факта обладания общим предварительно распределенным секретом (*PSK*). Аутентификация сервера является обязательной, аутентификация клиента является опциональной;
- конфиденциальность и целостность: обеспечивается за счет использования AEAD алгоритма.

Протокол TLS содержит два основных подпротокола, отвечающих за обеспечение свойств, перечисленных выше:

- протокол Handshake (раздел 5), отвечающий за согласование криптографических параметров, выработку общего ключевого материала и аутентификацию сторон;
- протокол Record (раздел 6), использующий криптографические параметры и общий ключевой материал, согласованные во время выполнения протокола Handshake, для защиты трафика между сторонами взаимодействия.

Стороны могут обмениваться информацией о закрытии соединения или о возникшей ошибке с помощью передачи соответствующих оповещений в рамках протокола Alert (раздел 7).

4.1 Иерархия информационного обмена

Протокол TLS 1.3 работает поверх транспортного протокола (например, TCP) с гарантированной доставкой пакетов данных, который обеспечивает доставку сообщений с сохранением их очередности, отсутствием потерь и дублирований. Поверх протокола TLS 1.3, в свою очередь, работают протоколы прикладного уровня.

Схема обмена данными в протоколе TLS 1.3 изображена на рисунке 1.

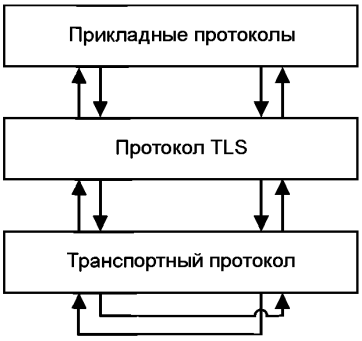


Рисунок 1 — Схема обмена данными в протоколе TLS 1.3

Иерархия информационного обмена протокола TLS 1.3 включает в себя соединения, в рамках которых пересылается поток сообщений различных типов, инкапсулированный в записи.

4.2 Состояние соединения

Состояние соединения определяет порядок обработки данных, передаваемых в рамках этого соединения. В каждый момент времени выделяется текущее состояние чтения и текущее состояние записи для каждой из сторон взаимодействия.

Для каждой из сторон с состоянием чтения/записи связаны следующие параметры:

- номер получаемой/пересылаемой записи *seqnum* (число от 0 до *SNMAX*-1 включительно, где параметр *SNMAX* задается выбранным криптонабором, см. 10.1.3), который увеличивается на единицу после каждой полученной/отправленной записи;
- ключевой материал трафика: [*sender*]_*write_key*, [*sender*]_*write_iv* (см. 8.4).

В каждый момент времени все записи обрабатываются в соответствии с текущим состоянием соединения. При инициализации соединения номеру записи присваивается нулевое значение, ключевой материал не определен (записи передаются в открытом виде). Смена состояния чтения/записи происходит при каждом изменении ключевого материала трафика, при этом соответствующий данному состоянию номер пересылаемой/получаемой записи обнуляется.

В рамках настоящих рекомендаций выделяется три этапа состояний соединения:

- этап ключевого обмена: сообщения ClientHello, HelloRetryRequest, ServerHello протокола Handshake (см. 5.5), всегда пересылаемые сторонами в открытом виде;
- этап выработки параметров соединения и аутентификации: все сообщения протокола Handshake, начиная с сообщения EncryptedExtensions и заканчивая сообщением Finished со стороны клиента, всегда передаются в защищенном виде с помощью ключей, выработанных на основе секретного значения [*sender*]_*handshake_traffic_secret* (см. 8.2);
- этап пересылки прикладных данных (см. раздел 9) и post-handshake сообщений (см. 5.9): все данные, посылаемые в рамках этого этапа, передаются в защищенном виде. Для их защиты используются ключи, выработанные на основе секретного значения [*sender*]_*application_traffic_secret_N* (см. 8.2).

Примечания

1 Прикладные данные, пересылаемые в защищенном на ключах [*sender*]_*application_traffic_secret_N* виде, могут быть посланы сервером до получения сообщения Finished со стороны клиента в случае односторонней аутентификации (см. подробнее раздел 9).

2 В рамках этапа пересылки прикладных данных может проходить смена состояния соединения путем смены ключевого материала трафика с помощью механизма сообщений KeyUpdate (см. 5.9.3).

5 Протокол Handshake

5.1 Формат сообщений протокола Handshake

Протокол Handshake используется для согласования параметров безопасности соединения, выработки общего ключевого материала и аутентификации сторон. Сообщения протокола Handshake передаются протоколу Record для последующей инкапсуляции в одну или несколько TLSPlaintext или

TLSCiphertext структур (см. раздел 6), которые обрабатываются и передаются в канале связи в соответствии с текущим состоянием соединения (см. 4.2).

Каждое сообщение протокола Handshake содержит специальный заголовок, состоящий из четырех байтов. Первый байт содержит код типа сообщения (поле msg_type), три следующих байта — длину данных сообщения (поле length). После заголовка следуют пересылаемые в данном сообщении данные.

```
enum {
    client_hello(0x01),
    server_hello(0x02),
    new_session_ticket(0x04),
    end_of_early_data(0x05),
    encrypted_extensions(0x08),
    certificate(0x0B),
    certificate_request(0x0D),
    certificate_verify(0x0F),
    finished(0x14),
    key_update(0x18),
    message_hash(0xFE),
    (0xFF)
} HandshakeType;

struct {
    HandshakeType msg_type; /* handshake type */
    uint24 length; /* remaining bytes in message */
    select (Handshake.msg_type) {
        case client_hello:          ClientHello;
        case server_hello:          ServerHello;
        case end_of_early_data:     EndOfEarlyData;
        case encrypted_extensions:  EncryptedExtensions;
        case certificate_request:   CertificateRequest;
        case certificate:           Certificate;
        case certificate_verify:    CertificateVerify;
        case finished:              Finished;
        case new_session_ticket:    NewSessionTicket;
        case key_update:            KeyUpdate;
    };
} Handshake;
```

Далее по тексту установлено, что под терминами «сообщение ClientHello», «сообщение ServerHello», ..., «сообщение KeyUpdate» подразумевается набор данных, соответствующих структуре Handshake (в частности, имеющий поля Handshake.msg_type и Handshake.length), описанной выше, для которых поле Handshake.msg_type содержит значения client_hello(0x01), server_hello(0x02), ..., key_update(0x18) соответственно.

Далее по тексту установлено, что строки *ClientHello*, *ServerHello*, ..., *KeyUpdate* являются байтовыми представлениями сообщений ClientHello, ..., KeyUpdate соответственно (см. 3.2).

Далее по тексту установлено, что под термином «первое сообщение Finished со стороны клиента» подразумевается сообщение Finished, посылаемое впервые в рамках текущего соединения.

Далее по тексту установлено, что под термином «main-handshake сообщения» подразумеваются сообщения, посылаемые сторонами взаимодействия в рамках работы протокола Handshake, начиная с исходного сообщения ClientHello и заканчивая первым сообщением Finished со стороны клиента, при этом сообщения NewSessionTicket и KeyUpdate, которые могут быть посланы сервером до получения первого сообщения Finished со стороны клиента в случае односторонней аутентификации (см. подробнее 5.9.1 и 5.9.3), не включаются в множество main-handshake сообщений.

Далее по тексту установлено, что под термином «post-handshake сообщения» подразумеваются все сообщения, посылаемые сторонами взаимодействия в рамках работы протокола Handshake, которые не являются main-handshake сообщениями.

Сообщения протокола Handshake должны пересылаться в строгом фиксированном порядке, который определяется следующим образом: ClientHello/(ClientHello1, HelloRetryRequest, ClientHello2), ServerHello, EncryptedExtensions, CertificateRequest, Certificate со стороны сервера, CertificateVerify со стороны сервера, Finished со стороны сервера, Certificate со стороны клиента, CertificateVerify со стороны клиента, Finished со стороны клиента, post-handshake сообщения (см. 5.9). При этом опциональные сообщения из данного списка могут опускаться. Сторона взаимодействия, получившая сообщение протокола Handshake, которое не соответствует данному порядку, должна прервать работу протокола с оповещением об ошибке unexpected_message (см. 7.2).

Примечание — В настоящих рекомендациях не описывается сообщение EndOfEarlyData, указанное в [1], так как в версии протокола TLS 1.3, соответствующей настоящим рекомендациям, пересылка 0-RTT данных запрещена.

5.2 Базовые принципы работы протокола Handshake

В рамках протокола TLS 1.3 существует два типа энтропийных данных, которые могут быть использованы для выработки общих секретных значений в соответствии с иерархией ключей (см. 8.2):

а) общее секретное значение *ECDHE*, вырабатываемое при использовании протокола Диффи-Хеллмана на основе эллиптических кривых[1)] в соответствии с 8.5. Для выработки данного значения стороны взаимодействия обмениваются открытыми эфемерными ключами в рамках сообщений ключевого обмена;

б) предварительно распределенный секрет, известный обеим сторонам до начала работы текущего соединения и выработанный в соответствии с одним из следующих способов (см. подробнее 8.6):

1) выработан в рамках работы протокола Handshake — внутренний предварительно распределенный секрет *iPSK*;

2) распределен между сторонами вне протокола TLS 1.3 — внешний предварительно распределенный секрет *ePSK*. Настоящие рекомендации не фиксируют механизм формирования и распределения значения *ePSR*. При необходимости использования данного типа предварительно распределенного секрета описание данного механизма, исследование предоставляемого посредством использования данного значения *ePSK* функционала, а также анализ стойкости протокола должны проводиться отдельно.

В рамках протокола TLS 1.3 существует два способа аутентификации сторон:

а) аутентификация за счет использования сертификатов и подписи. При этом аутентификация сервера является обязательной и происходит только в рамках пересылки соответствующих main-handshake сообщений, а аутентификация клиента является опциональной, выполняется по запросу от сервера и проходит в рамках пересылки как main-handshake, так и post-handshake сообщений (см. 5.9.2);

б) аутентификация за счет подтверждения (неявного) факта обладания предварительно распределенным секретом. При этом тип аутентификации (двусторонняя или односторонняя) либо определяется типом аутентификации сторон в соединении на момент пересылки сообщения NewSessionTicket, ассоциированного с используемым внутренним предварительно распределенным секретом *iPSK*, либо всегда является двусторонней[2)] в случае использования внешнего предварительно распределенного секрета *ePSK*.

[1)] В соответствии с [1] протокол TLS 1.3 допускает возможность работы протокола Handshake на основе использования протокола Диффи-Хеллмана в мультипликативной группе конечного поля, однако этот способ в настоящих рекомендациях не поддерживается.

[2)] Данное требование отсутствует в [1] и является дополнительным условием, накладываемым на внешний предварительно распределенный секрет в рамках настоящих рекомендаций.

Примечания

1 Далее по тексту под «сертификатом сервера, ассоциированным со значением *iPSK*», подразумевается сертификат, который был использован сервером для успешной аутентификации при установлении первоначального соединения из цепочки соединений, в рамках которой было выработано данное значение *iPSK*. Под «сертификатом клиента, ассоциированным со значением *iPSK*», подразумевается сертификат, который был использован клиентом для успешной аутентификации при установлении некоторого соединения из цепочки соединений, в рамках которой было выработано данное значение *iPSK*, и который определяет состояние аутентификации клиента в соединении на момент пересылки сообщения NewSessionTicket, ассоциированного с данным значением *iPSK*.

2 Далее по тексту установлено, что под термином «значение *ePSK*, ассоциированное со значением *iPSK*», подразумевается значение *ePSK*, согласованное в рамках первоначального соединения из цепочки соединений, в результате которой было выработано данное значение *iPSK*.

В настоящих рекомендациях описываются три схемы аутентифицированной выработки общего ключевого материала, используемые в режимах работы протокола TLS 1.3:

а) ecdhe_ke (см. подробнее 5.3.1). В качестве энтропийных данных для выработки общего ключевого материала соединения используется общее секретное значение *ECDHE*, вырабатываемое сторонами на этапе пересылки сообщений ключевого обмена (см. 8.5). Аутентификация сторон осуществляется за счет использования сертификатов и подписи;

б) psk_ke (см. подробнее 5.3.2.1). В качестве энтропийных данных для выработки общего ключевого материала соединения используется общее секретное PSK-значение, вырабатываемое в соответствии с 8.6 (в рамках настоящих рекомендаций допускается использование только внутреннего предварительно распределенного секрета *iPSK*). Аутентификация сторон осуществляется за счет подтверждения (неявного) факта обладания PSK-значением;

в) psk_ecdhe_ke (см. подробнее 5.3.2.2): на основе использования двух подходов, перечисленных выше. В качестве энтропийных данных для выработки общего ключевого материала соединения используется предварительно распределенный секрет *PSK*, вырабатываемый в соответствии с 8.6, который может быть как внутренним (*iPSK*), так и внешним (*ePSK*), и общее секретное значение *ECDHE*, вырабатываемое сторонами на этапе пересылки сообщений ключевого обмена (см. 8.5). Аутентификация сторон осуществляется за счет подтверждения (неявного) факта обладания значением *PSK*.

Режимы работы протокола Handshake подразделяются на два следующих типа:

а) полная схема обмена сообщениями (Full Handshake): к данному типу относятся ECDHE-only и ePSK-ECDHE режимы;

б) возобновление соединения (Resumption): к данному типу относятся iPSK-only и iPSK-ECDHE режимы. Данные режимы работы являются аналогами механизма возобновления соединения за счет использования идентификаторов сессии, применяемого в более ранних версиях протокола TLS (см. Р 1323565.1.020), и используются для быстрого восстановления параметров с помощью внутреннего предварительно распределенного секрета *iPSK*.

В таблице 1 приводится информация о соответствии режимов работы протокола Handshake схемам аутентифицированной выработки общего ключевого материала и типам используемых энтропийных данных.

Таблица 1 — Соответствие режимов работы протокола Handshake схемам аутентифицированной выработки общего ключевого материала

Режим работы протокола Handshake	Схема аутентифицированной выработки общего ключевого материала	Используемые энтропийные данные
ECDHE-only	ecdhe_ke	*ECDHE*
ePSK-ECDHE	psk_ecdhe_ke	*ePSK и ECDHE*
iPSK-only	psk_ke	*iPSK*
iPSK-ECDHE	psk_ecdhe_ke	*iPSK* и *ECDHE*

Соотношение режимов работы протокола Handshake приведено на рисунке 2. Основной принцип заключается в следующем: внешне распределенный секрет *ePSK* может быть использован только в рамках ePSK-ECDHE режима.

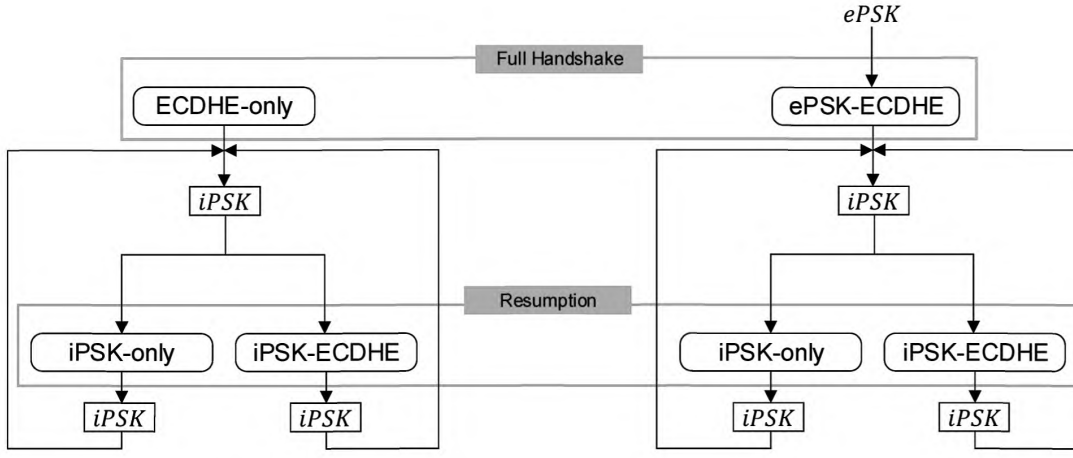


Рисунок 2 — Соотношение режимов работы протокола Handshake

Примечание — В соответствии с [1] протокол TLS 1.3 допускает использование 0-RTT данных, однако при работе протокола в соответствии с настоящими рекомендациями, данный функционал запрещен.

5.3 Схемы аутентифицированной выработки общего ключевого материала

В настоящем разделе описываются три схемы аутентифицированной выработки общего ключевого материала ecdhe_ke, psk_ecdhe_ke, psk_ke, для каждой из которых приводится схема режимов работы протокола Handshake (см. рисунки 3—5), содержащая список всех сообщений протокола Handshake, которые могут быть посланы в рамках данного режима [кроме сообщения HelloRetryRequest, которое может посылаться в рамках ecdhe_ke и psk_ecdhe_ke режимов (см. подробнее 5.4 и 5.5.3)], а также базовый набор расширений, непосредственно используемых для аутентифицированной выработки общего ключевого материала в каждом конкретном режиме.

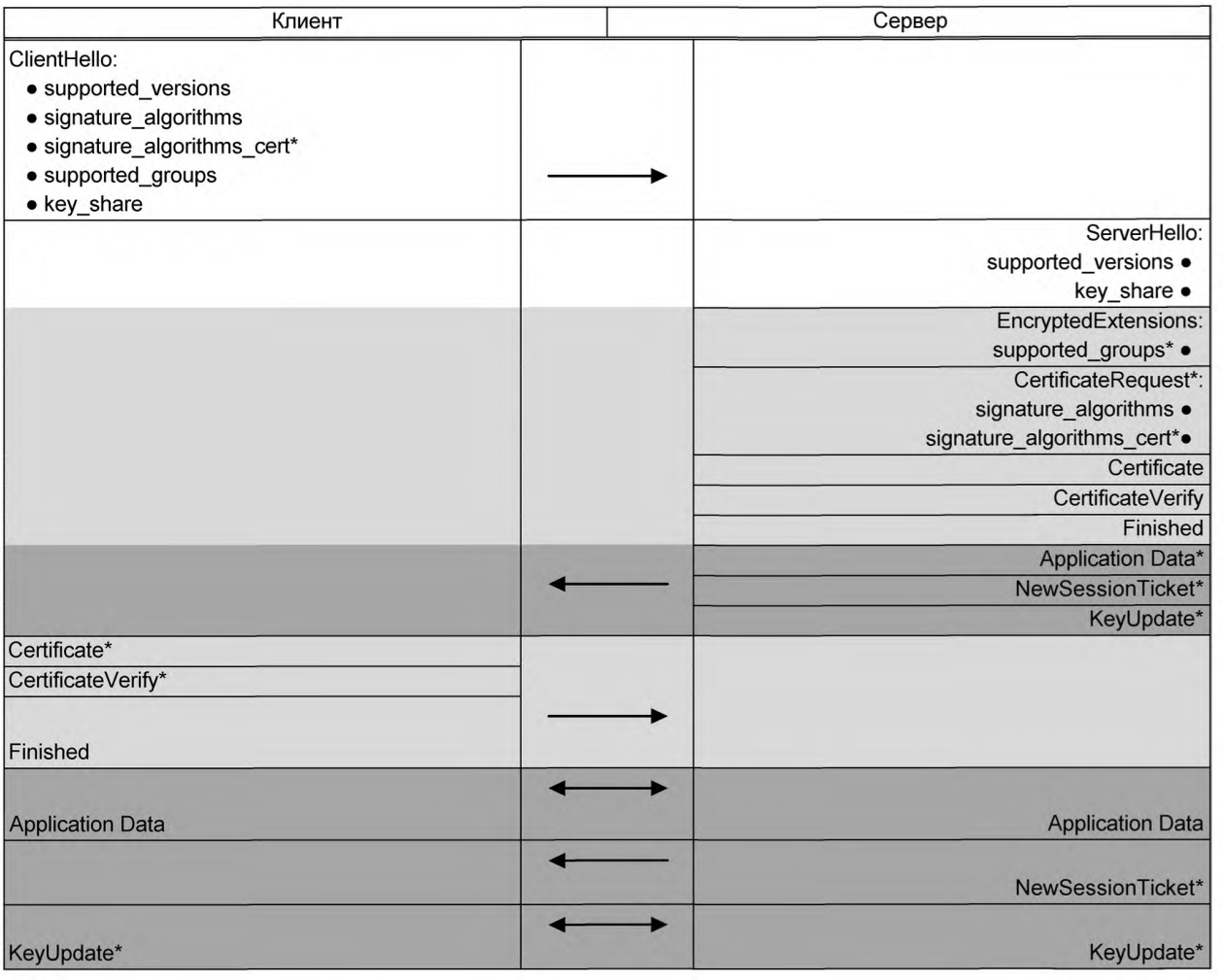
Полный список всех расширений приведен в таблице 2. Подробное описание каждого из сообщений приводится в 5.5—5.9.

5.3.1 Схема ecdhe_ke

Схема аутентифицированной выработки общего ключа ecdhe_ke основывается на использовании протокола Диффи-Хеллмана на основе эллиптических кривых. В качестве энтропийных данных для выработки общего ключевого материала соединения используется общее секретное значение *ECDHE*, вырабатываемое сторонами на этапе пересылки сообщений ключевого обмена (см. 8.5). Аутентификация сторон происходит за счет использования сертификатов и подписи, пересылаемых в рамках сообщений Certificate и CertificateVerify (см. подробнее 5.8.1 и 5.8.2).

Схема ecdhe_ke используется в рамках ECDHE-only режима работы протокола Handshake.

Схема обмена сообщениями в ECDHE-only режиме работы протокола Handshake приведена на рисунке 3.



Используемая система обозначений:

•	– расширения, посылаемые в рамках сообщения, под которым они указаны;
*	– опциональные данные;
	– сообщения, защищенные на ключах, выработанных из секретного значения [*sender*]_*handshake*_*traffic*_*secret* (см. подробнее 8.4);
	– сообщения, защищенные на ключах, выработанных из секретного значения [*sender*]_*application*_*traffic*_*secret*_*N* (см. подробнее 8.4).

Рисунок 3 — Схема обмена сообщениями в ECDHE-only режиме работы протокола Handshake

Примечания

1 На рисунке 3 прикладные данные Application Data не относятся к сообщениям протокола Handshake.

2 Прикладные данные (Application Data), пересылаемые сервером до получения первого сообщения Finished со стороны клиента, могут быть посланы только в случае односторонней аутентификации (см. подробнее раздел 9).

3 Сообщения NewSessionTicket и KeyUpdate, пересылаемые сервером до получения первого сообщения Finished со стороны клиента, могут быть посланы только в случае односторонней аутентификации (см. подробнее 5.9.1, 5.9.3).

Для установления соединения в рамках ecdhe_ke схемы в сообщении ClientHello необходимо указать следующий минимальный набор расширений:

а) обязательное расширение supported_versions, используемое для согласования версии протокола TLS (см. 5.6.1);

б) обязательное расширение signature_algorithms, содержащее информацию об алгоритмах подписи, поддерживаемых клиентом, и отвечающее за возможность аутентификации сторон с помощью сертификатов (см. 5.6.2);

в) обязательные расширения supported_groups (см. 5.6.3) и key_share (см. 5.6.4), отвечающие за выработку общего секретного значения *ECDHE*, где:

1) расширение supported_groups содержит информацию об эллиптических кривых, поддерживаемых клиентом и указываемых в порядке убывания предпочтения;

2) расширение key_share содержит информацию об открытых эфемерных ключах Q^1_c, Q^2_c, ... , предлагаемых клиентом и указываемых в порядке убывания предпочтения.

Для установления соединения в рамках ecdhe_ke схемы в сообщении ServerHello необходимо указать следующий минимальный набор расширений:

а) обязательное расширение supported_versions, используемое для согласования версии протокола TLS (см. 5.6.1);

б) обязательное расширение key_share, содержащее информацию об открытом эфемерном ключе сервера Q_S, принадлежащему той же кривой, что и некоторый открытый эфемерный ключ клиента Q^i_c, выбранный сервером из списка, указанного в расширении key_share со стороны клиента. При этом данное расширение посылается, в случае если сервер готов работать с данными, переданными в расширениях supported_groups и key_share со стороны клиента.

Примечания

1 Расширение supported_groups является опциональным для сервера и содержит информацию об эллиптических кривых, поддерживаемых сервером.

2 Расширение signature_algorithms является обязательным для сервера в случае двусторонней аутентификации и должно пересылаться в сообщении CertificateRequest.

3 Расширение signature_algorithms_cert является опциональным для клиента и сервера (см. подробнее 5.6.2).

После обмена сообщениями ключевого обмена стороны вырабатывают общее секретное значение *ECDHE* в соответствии с 8.5. Данное значение используется в качестве энтропийных данных для формирования иерархии ключей (см. 8.2).

Сервер может воспользоваться процедурой пересогласования открытых эфемерных ключей клиента (см. 5.4). Если какая-либо из сторон не может использовать параметры, предложенные второй стороной, то она должна завершить соединение либо с оповещением handshake_failure, либо с оповещением insufficient_security (см. 7.2).

В случае необходимости смены ключевого материала трафика любая из сторон может послать сообщение KeyUpdate (см. 5.9.3).

5.3.2 Схемы, использующие предварительно распределенный секрет

Для установления защищенного соединения между клиентом и сервером может быть использовано PSK-значение, известное обеим сторонам до начала работы текущего соединения и выработанное в соответствии с одним из следующих способов:

а) выработано в рамках работы протокола Handshake в соединении, предшествующем текущему (инициализирующем соединении), в соответствии с 8.6: внутренний предварительно распределенный секрет *iPSK*. При этом значение *iPSK* может быть использовано в рамках работы iPSK-only и iPSK-ECDHE режимов;

б) распределено между сторонами вне протокола TLS 1.3: внешний предварительно распределенный секрет *ePSK*. При этом значение *ePSK* должно обеспечивать двустороннюю аутентификацию сторон и может быть использовано только в рамках работы ePSK-ECDHE режима.

Двусторонняя аутентификация с помощью использования внешнего предварительно распределенного секрета *ePSK* обеспечивается с помощью механизма внешнего распределения секрета между двумя сторонами, гарантирующего, что данный секрет неизвестен никому, кроме данных сторон. При этом в рамках данного механизма должны быть однозначно зафиксированы роли сторон взаимодействия (клиент или сервер), причем каждой стороне может соответствовать только одна роль на протяжении всего срока жизни *ePSK*.

Примечания

1 Требование двусторонней аутентификации отсутствует в [1] и является дополнительным условием, накладываемым на внешний предварительно распределенный секрет *ePSK* в рамках настоящих рекомендаций.

2 Значение *iPSK* не рекомендуется использовать в качестве энтропийных данных для аутентифицированной выработки общего ключевого материала более чем для одного соединения.

Для выработки значения *iPSK* в рамках работы инициализирующего соединения сервер может послать сообщение NewSessionTicket (см. подробнее 5.9.1), содержащее данные, однозначно ассоциированные с секретным значением *iPSK* (см. подробнее 8.6).

Аутентификация сторон в рамках psk_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала всегда происходит за счет подтверждения (неявного) факта обладания PSK-значением, при этом сообщения CertificateRequest, Certificate и CertificateVerify посылаться не должны.

Для установления соединения в рамках psk_ke или psk_ecdhe_ke схем в сообщении ClientHello необходимо указать следующий минимальный набор расширений:

а) обязательное расширение supported_versions, используемое для согласования версии протокола TLS (см. 5.6.1);

б) обязательные для каждой из двух схем (psk_ke или psk_ecdhe_ke) расширения pre_shared_key (см. 5.6.5) и psk_key_exchange_modes (см. 5.6.6), отвечающие за согласование PSK-значения и режима его использования:

1) расширение pre_shared_key содержит список данных, связанных со значениями предварительно распределенных секретов, которые клиент готов использовать в качестве энтропийных данных;

2) расширение psk_key_exchange_modes, содержащее информацию о схемах аутентифицированной выработки общего ключевого материала, поддерживаемых клиентом;

в) обязательные для psk_ecdhe_ke схемы расширения key_share и supported_groups, отвечающие за выработку общего секретного значения *ECDHE* и посылаемые клиентом в случае, если он готов поддерживать данный функционал.

Примечание — Чтобы предоставить серверу возможность при необходимости отклонить процедуру возобновления соединения, клиенту рекомендуется всегда посылать расширения, обязательные для установления соединений в соответствии с полной схемой обмена сообщениями.

Для установления соединения в рамках psk_ke или psk_ecdhe_ke схем в сообщении ServerHello необходимо указать следующий минимальный набор расширений:

а) обязательное расширение supported_versions, используемое для согласования версии протокола TLS (см. 5.6.1);

б) обязательное для каждой из двух схем (psk_ke или psk_ecdhe_ke) расширение pre_shared_key, содержащее информацию о выбранном сервером значении *PSK* и посылаемое в случае, если сервер готов работать с данными, переданными в расширениях pre_shared_key и psk_key_exchange_modes со стороны клиента;

в) обязательное для psk_ecdhe_ke схемы расширение key_share, отвечающее за выработку общего секретного значения *ECDHE*. В случае psk_ke схемы данное расширение посылаться не должно.

Примечание — Расширение supported_groups является опциональным для сервера и содержит информацию об эллиптических кривых, поддерживаемых сервером.

После обмена сообщениями ключевого обмена стороны взаимодействия согласовывают общее значение *PSK* и, опционально, вырабатывают общее секретное значение *ECDHE* (см. подробнее 8.5), используемые в качестве энтропийных данных для формирования иерархии ключей (см. 8.2).

Если какая-либо из сторон не может работать на параметрах, предложенных второй стороной, то она должна завершить соединение либо с оповещением handshake_failure, либо с оповещением insufficient_security (см. 7.2).

Примечание — Если при возобновлении соединения сервер не может согласовать параметры, пересылаемые в расширениях pre_shared_key и psk_key_exchange_modes со стороны клиента, но при этом клиент предоставил набор расширений, необходимый для перехода на полную схему обмена сообщениями, сервер может не разрывать соединение и перейти на полную схему обмена сообщениями, указав соответствующий ответный набор расширений.

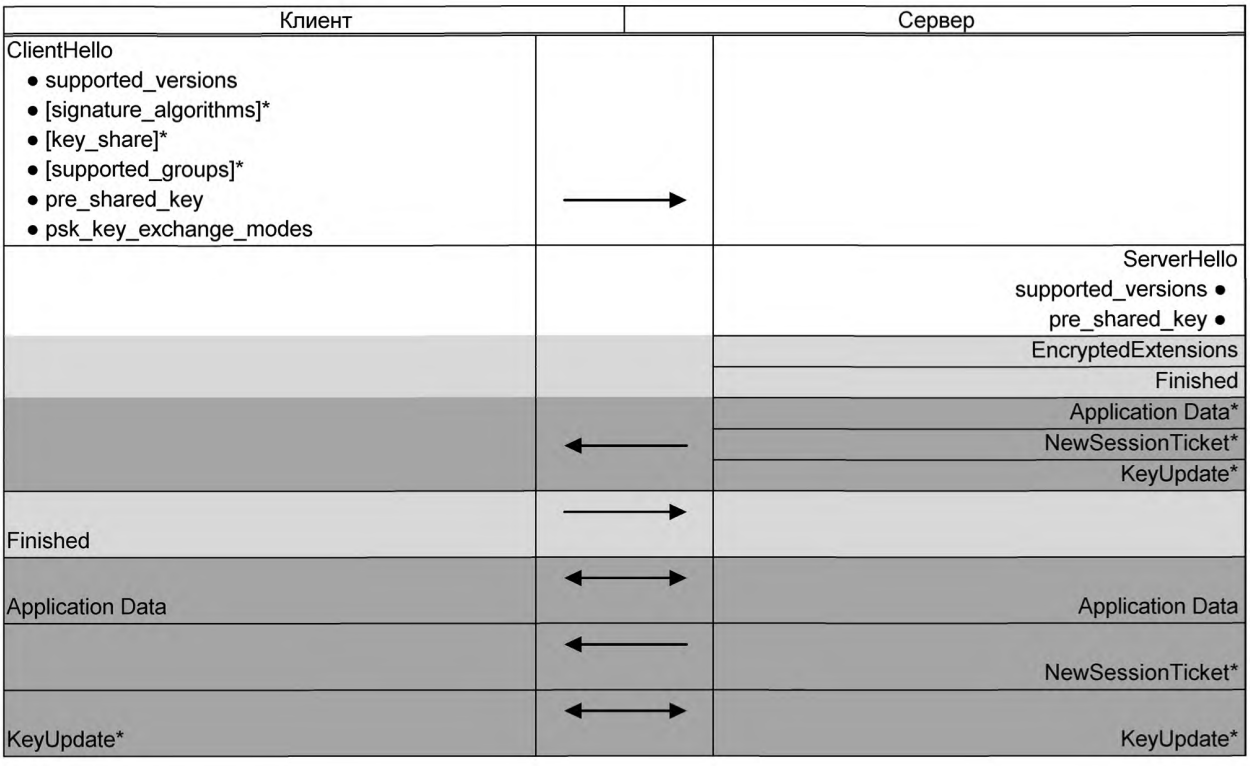
В случае необходимости смены ключевого материала трафика любая из сторон может послать сообщение KeyUpdate (см. 5.9.3).

Подробная схема режимов работы протокола Handshake в рамках psk_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала приводится в 5.3.2.1 и 5.3.2.2 соответственно.

Примечание — В соответствии с [1] протокол TLS 1.3 допускает использование 0-RTT данных, однако при работе протокола в соответствии с настоящими рекомендациями данный функционал запрещен.

5.3.2.1 Схема psk_ke

Схема psk_ke используется в рамках iPSK-only режима работы протокола Handshake, схема работы которого приведена на рисунке 4.



Используемая система обозначений:

•	– расширения, посылаемые в рамках сообщения, под которым они указаны;
*	– опциональные данные;
[]	– расширение, посылаемое клиентом в режиме возобновления соединения для обеспечения возможности перехода к полной схеме обмена сообщениями;
	– сообщения, защищенные на ключах, выработанных из секретного значения [*sender*]_*handshake_traffic_secret* (см. подробнее 8.4);
	– сообщения, защищенные на ключах, выработанных из секретного значения [*sender*]_*application_traffic_secret_N* (см. подробнее 8.4).

Рисунок 4 — Схема обмена сообщениями в iPSK-only режиме работы протокола Handshake

Примечания

1 Прикладные данные Application Data не относятся к сообщениям протокола Handshake.

2 На рисунке 4 прикладные данные Application Data, пересылаемые сервером до получения первого сообщения Finished со стороны клиента, могут быть посланы только в случае односторонней аутентификации (подробнее раздел 9).

3 Сообщения NewSessionTicket и KeyUpdate, пересылаемые сервером до получения первого сообщения Finished со стороны клиента, могут быть посланы только в случае односторонней аутентификации (см. подробнее 5.9.1, 5.9.3).

5.3.2.2 Схема psk_ecdhe_ke

Схема psk_ecdhe_ke используется в рамках ePSK-ECDHE или iPSK-ECDHE режимов работы протокола Handshake, схема работы которых приведена на рисунке 5.

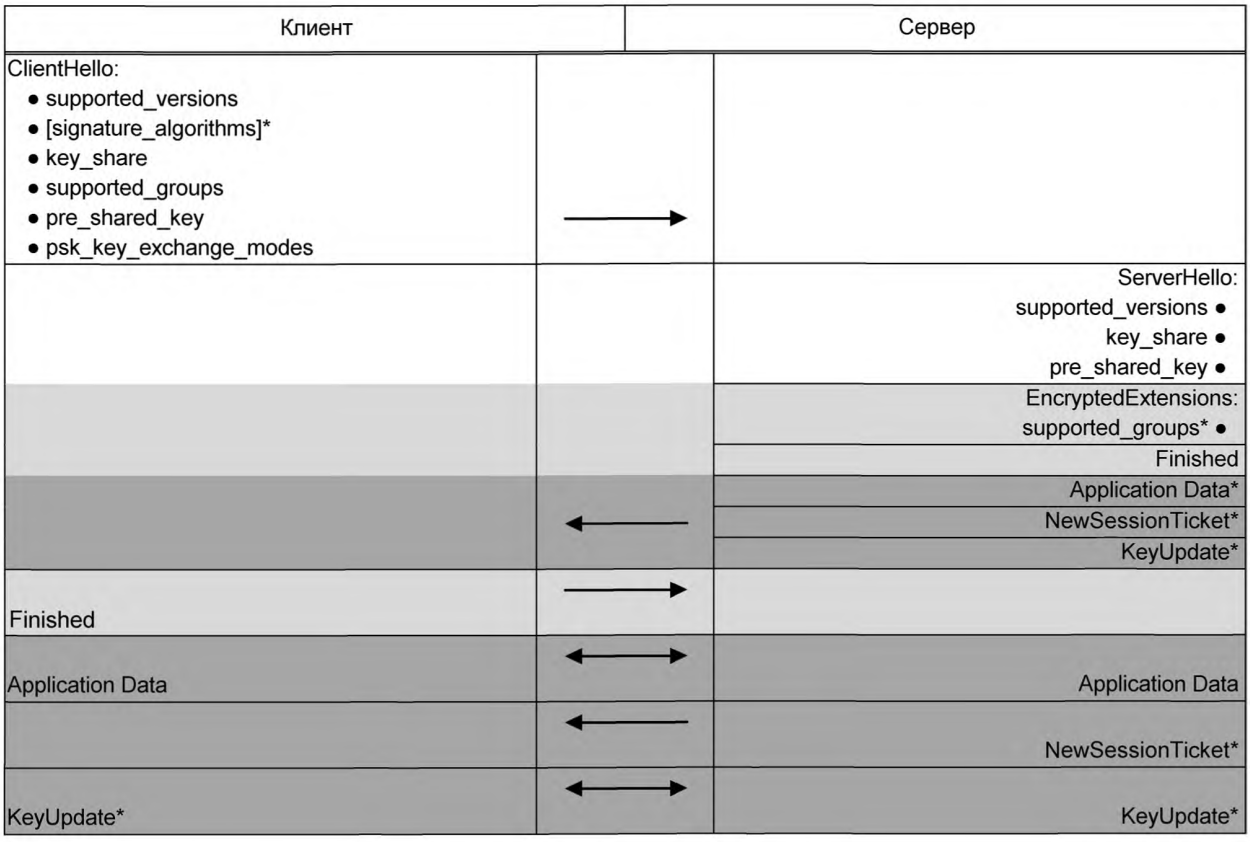


Рисунок 5 — Схема обмена сообщениями в ePSK-ECDHE и iPSK-ECDHE режимах работы протокола Handshake

Примечания

1 Прикладные данные Application Data не относятся к сообщениям протокола Handshake.

2 На рисунке 5 прикладные данные Application Data, пересылаемые сервером до получения первого сообщения Finished со стороны клиента, могут быть посланы только в случае односторонней аутентификации (подробнее раздел 9).

3 Сообщения NewSessionTicket и KeyUpdate, пересылаемые сервером до получения первого сообщения Finished со стороны клиента, могут быть посланы только в случае односторонней аутентификации (см. подробнее 5.9.1, 5.9.3).

В случае если сервер не поддерживает параметры, указанные в расширении key_share, он может воспользоваться процедурой пересогласования открытых эфемерных ключей клиента (см. 5.4).

5.4 Пересогласование открытых эфемерных ключей

Если в рамках режимов на основе ecdhe_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала данные, переданные клиентом в расширении key_share сообщения ClientHello1, не могут быть согласованы сервером (например, соответствуют параметрам эллиптиче-

ских кривых, не поддерживаемых сервером), но в расширении supported_groups указаны параметры, поддерживаемые сервером, сервер может инициировать процедуру пересогласования открытых эфемерных ключей клиента. Для этого он должен послать клиенту сообщение HelloRetryRequest (см. 5.5.3).

При получении сообщения HelloRetryRequest клиенту необходимо послать модифицированное сообщение ClientHello2, содержащее расширение key_share с параметрами, скорректированными в соответствии с полученным ранее сообщением HelloRetryRequest.

Примечание — Обозначения «ClientHello1», «ClientHello2», используемые в настоящем разделе, подробно объясняются в 5.5.1.

5.5 Сообщения ключевого обмена

5.5.1 ClientHello

Сообщение приветствия ClientHello посылается клиентом в одном из следующих случаев:

- клиент впервые подключается к серверу (исходное сообщение ClientHello);
- в ответ на сообщение HelloRetryRequest, посланное сервером (повторное сообщение ClientHello).

Примечание — Далее по тексту установлено, что под терминами «сообщение ClientHello1», «сообщение ClientHello2» подразумеваются исходное и повторное сообщение соответственно, а строки ClientHello1, ClientHello2 являются байтовыми представлениями сообщений ClientHello1, ClientHello2 соответственно.

При получении сообщения ClientHello в любом другом случае, не перечисленном выше, сервер должен завершить соединение оповещением unexpected_message(см. 7.2), поскольку версия протокола, описанная в настоящих рекомендациях, не поддерживает процедуру пересогласования соединения (renegotiation), определенную в Р 1323565.1.020.

В случае если сервер устанавливает соединение в рамках протокола TLS версии ниже 1.3 и получает в рамках процедуры пересогласования соединения (renegotiation) сообщение ClientHello, сформированное в соответствии с протоколом TLS 1.3, он должен сохранить предыдущую версию протокола TLS. В частности, сервер не должен согласовывать протокол TLS версии 1.3 в указанном случае.

В случае получения сообщения HelloRetryRequest клиент должен отправить сообщение ClientHello2, содержащее следующие изменения по сравнению с сообщением ClientHello1:

- если расширение key_share было указано в сообщении HelloRetryRequest, список эфемерных ключей KeyShareClientHello должен содержать один элемент KeyShareEntry, соответствующий группе эллиптической кривой, указанной в расширении key_share сообщения HelloRetryRequest (см. 5.6.4.2);
- если расширение cookie(см. 5.6.8) было указано в сообщении HelloRetryRequest, оно должно присутствовать в сообщении ClientHello2;
- расширение pre_shared_key, если оно присутствовало в сообщении ClientHello1, должно быть обновлено в сообщении ClientHello2 путем перевычисления поля obfuscated_ticket_age структуры PskIdentity в структуре PreSharedKeyExtension, описанной в 5.6.5, и binder-значений, описанных в 5.6.5.3; а также путем (опционального) удаления тикетов (и всей связанной с данными тикетами информации), несовместимых с криптонабором, указанным сервером;
- могут быть проведены модификации других опциональных расширений, определенных вне данных рекомендаций и присутствующих в сообщении HelloRetryRequest, однако возможность таких модификаций должна оговариваться отдельно и не рассматривается в рамках текущего документа.

Структура ClientHello сообщения ClientHello задается следующим образом:

```
uint16 ProtocolVersion;
opaque Random[32];
uint8 CipherSuite[2];       /* Cryptographic suite selector */
struct {
    ProtocolVersion legacy_version = 0x0303;     /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;
```

где legacy_version — поле длиной в 2 байта, отвечающее в более ранних версиях протокола TLS за индикацию максимальной версии протокола, поддерживаемой клиентом, и сохраненное в текущей версии протокола в целях поддержки совместимости форматов сообщений.

В протоколе TLS 1.3 информация о поддерживаемых клиентом версиях указывается в расширении supported_versions, описанном в 5.6.1.

В рамках протокола TLS 1.3 данное поле должно принимать значение 0x0303 (соответствующее значению поля client_version сообщения ClientHello для протокола TLS 1.2);

random — строка данных длиной в 32 байта, выработанная клиентом случайным образом;

legacy_session_id — поле, отвечающее в более ранних версиях протокола TLS за механизм возобновления соединения, определенный в Р 1323565.1.020, замененный в настоящих рекомендациях механизмом использования значений *iPSK*, и сохраненное в текущей версии протокола в целях поддержки совместимости форматов сообщений.

В рамках работы протокола TLS 1.3 не в режиме совместимости данное поле должно содержать вектор нулевой длины.

В рамках работы протокола TLS 1.3 в режиме совместимости данное поле должно содержать строку данных длиной в 32 байта, сгенерированную клиентом. При этом данное значение не обязательно должно быть случайным, но должно быть непредсказуемым (таким, что его нельзя заранее определить с достаточно большой вероятностью);

cipher_suites — список криптонаборов, которые поддерживает клиент. Порядок криптонаборов в списке отражает их степень предпочтения (предпочтительные идут первыми). Если список содержит криптонаборы, которые сервер не распознает, не поддерживает или не желает использовать, сервер должен их проигнорировать. Если клиент пытается установить соединение в рамках psk_ecdhe_ke или psk_ke схемы аутентифицированной выработки общего ключевого материала, ему следует предъявить по меньшей мере один криптонабор, поддерживающий алгоритм хэширования, ассоциированный с предложенными тикетами (см. 5.6.5.2).

Значения криптонаборов, допустимых к использованию в рамках данного документа, задаются в 10.1;

legacy_compression_methods — поле, содержащее вектор длиной 1 со значением, равным 0 (соответствующее методу null), отвечавшее в более ранних версиях протокола TLS за выбор метода сжатия, использование которого запрещено в рамках текущей версии протокола. Если в полученном сообщении ClientHello значение этого поля отлично от нуля, сервер должен прекратить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

В режиме совместимости сервер может получать сообщения ClientHello протокола TLS версии 1.2 и ниже, содержащие методы сжатия. В этом случае сервер должен следовать процедурам, соответствующим указанной версии TLS;

extensions — расширения, посылаемые со стороны клиента. Расширения выбираются из списка, приведенного в таблице 2. Серверы должны игнорировать нераспознанные расширения.

Можно выделить три типа расширений: расширение supported_versions, необходимое для согласования версии протокола (см. 5.6.1); расширения, необходимые для обеспечения корректной работы протокола Handshake, использующего одну из трех схем аутентифицированной выработки общего ключевого материала: psk_ke, psk_ec-

dhe_ke или ecdhe_ke, описанную в 5.3.2 и 5.3.1 соответственно; а также опциональные расширения, отвечающие за использование дополнительного функционала протокола.

В случае если сервер отказался поддерживать какое-либо из расширений, предложенных клиентом, клиент может прекратить работу протокола Handshake, послав соответствующее оповещение (например, missing_extension, см. подробнее 7.2).

В случае если клиент указывает значение 0x0304 в расширении supported_versions (см. 5.6.1), сообщение ClientHello должно удовлетворять следующим условиям:

- если указано расширение pre_shared_key, данное сообщение должно содержать расширение psk_key_exchange_modes (см. 5.6.6);
- если не указано расширение pre_shared_key, данное сообщение должно содержать расширения signature_algorithms и supported_groups;
- если указано расширение supported_groups, данное сообщение должно содержать расширение key_share, и наоборот. При этом список client_shares, указываемый клиентом в расширении key_share, может быть пустым (см. 5.6.4.1).

Сервер, получивший сообщение ClientHello, которое не удовлетворяет указанным условиям, должен завершить работу протокола Handshake с оповещением missing_extension(см. 7.2).

После отправки сообщения ClientHello клиент ожидает от сервера сообщения ServerHello или HelloRetryRequest.

5.5.2 ServerHello

Данное сообщение отправляется сервером после получения им сообщения ClientHello при условии, что среди параметров, переданных клиентом в сообщении приветствия, присутствует поддерживаемый сервером набор параметров, необходимый для продолжения установления соединения.

Структура ServerHello сообщения ServerHello задается следующим образом:

```
struct {
    ProtocolVersion legacy_version = 0x0303;      /* TLS v1.2 */
    Random random;
    opaque legacy_session_id_echo<0..32>;
    CipherSuite cipher_suite;
    uint8 legacy_compression_method = 0;
    Extension extensions<6..2^16-1>;
} ServerHello;
```

где legacy_version — поле длиной в 2 байта, отвечающее в более ранних версиях протокола TLS за индикацию версии протокола, поддерживаемой сервером, и сохраненное в текущей версии протокола в целях поддержки совместимости форматов сообщений.

В протоколе TLS 1.3 информация о выбранной сервером версии указывается в расширении supported_versions, описанном в 5.6.1.

В рамках протокола TLS 1.3 данное поле должно принимать значение 0x0303 (соответствующее значению поля server_version сообщения ServerHello для протокола TLS 1.2);

random — строка данных длиной в 32 байта, выработанная сервером случайным образом и не зависящая от значения, переданного клиентом в поле ClientHello.random.

В случае если TLS 1.3 сервер согласовывает параметры соединения в рамках режима совместимости и максимально поддерживаемая версия протокола TLS, указанная клиентом в сообщении ClientHello, соответствует версии, предшествующей версии 1.3, в целях защиты от downgrade атак сервер использует механизм формирования значения поля random в соответствии с 11.2.1;

legacy_session_id_echo — поле, содержащее данные, указанные клиентом в поле legacy_session_id сообщения ClientHello, и заполняемое таким образом даже в случае, если значение, указанное клиентом в поле legacy_session_id сообщения ClientHello, соответствует кэшированному значению сессии, соответствующей версии протокола TLS 1.2 и ниже.
Клиент, получивший сообщение ServerHello или HelloRetryRequest со значением поля legacy_session_id_echo, не соответствующим значению, посланному в сообщении ClientHello, должен прекратить работу протокола Handshake с оповещением illegal_parameter(см. 7.2);

cipher_suite — криптонабор, выбранный сервером из списка ClientHello.cipher_suites, предложенного клиентом. Клиент, получивший криптонабор, который не был предложен им в списке ClientHello.cipher_suites, должен прекратить работу протокола Handshake с оповещением illegal_parameter (см. 7.2);

legacy_compression_method — поле длиной в 1 байт, которое должно принимать значение 0 (соответствующее методу null);

extensions — список расширений. Сообщение ServerHello может содержать только те расширения, которые были перечислены в поле extensions сообщения ClientHello.
Сообщение ServerHello должно всегда содержать расширение supported_versions, необходимое для согласования версии протокола.
Сообщение ServerHello должно содержать набор расширений, необходимых для формирования криптографического контекста, соответствующего одной из трех схем аутентифицированной выработки общего ключевого материала psk_ke, psk_ecdhe_ke или ecdhe_ke, описанных в 5.3.2 и 5.3.1 соответственно, в рамках которой сервер планирует дальнейшее взаимодействие с клиентом. Таким образом, данное поле должно содержать либо расширение pre_shared_key (в случае использования psk_ke схемы), либо расширение key_share (в случае использования ecdhe_ke схемы), либо оба указанных расширения (в случае использования psk_ecdhe_ke схемы). Все остальные расширения должны пересылаться в сообщении EncryptedExtensions.

TLS 1.3 клиент, выполняющий процедуру пересогласования соединения в рамках протокола TLS версии 1.2 и ниже и получивший сообщение ServerHello, соответствующее протоколу TLS 1.3, в момент повторного согласования, должен прекратить работу протокола Handshake с оповещением protocol_version (см. 7.2).

5.5.3 HelloRetryRequest

Сервер может отправить сообщение HelloRetryRequest в ответ на сообщение ClientHello (далее — ClientHello1), в случае если сервер смог выбрать подходящие для работы параметры, но клиент не указал в сообщении ClientHello1 достаточного количества данных, необходимых для завершения этапа ключевого обмена. Например, клиент мог указать идентификатор поддерживаемой сервером эллиптической кривой в расширении supported_groups, но не указать соответствующий ей открытый эфемерный ключ в расширении key_share. Также сервер может отправить сообщение HelloRetryRequest для того, чтобы воспользоваться механизмом, предоставляемым расширением cookie (см. 5.6.8).

Сообщение HelloRetryRequest имеет точно такой же тип (server_hello, см. подробнее раздел 5) и структуру, что и сообщение ServerHello. При этом значения полей данного сообщения задаются в соответствии с теми же правилами, что и поля сообщения ServerHello, за исключением полей random и extensions:

а) поле random сообщения HelloRetryRequest должно задаваться строкой, содержащей значение хэш-функции SHA-256 от сообщения "HelloRetryRequest", равное

CF 21 AD 74 E5 9A 61 11 BE 1D 8C 02 1E 65 B8 91

C2 A2 11 16 7A BB 8C 5E 07 9E 09 E2 C8 A8 33 9C;

б) поле extensions сообщения HelloRetryRequest должно соответствовать следующим требованиям:

1) может содержать только те расширения, которые были перечислены в поле extensions сообщения ClientHello1, за исключением опционального расширения cookie(см. подробнее 5.6.8);

2) должно содержать расширение supported_versions и минимальный набор расширений, необходимый для того, чтобы клиент сгенерировал корректные параметры для выполнения этапа ключевого обмена.

После получения сообщения с типом server_hello клиент должен проверить значение поля random и, если это значение окажется равным фиксированной строке, указанной выше, выполнять действия в соответствии со случаем получения сообщения HelloRetryRequest (см. 5.5.1).

Если сообщение HelloRetryRequest не привело ни к одному изменению в сообщении ClientHello, клиент должен завершить работу протокола Handshake с оповещением illegal_parameter(см. 7.2). В случае повторного получения сообщения HelloRetryRequest в рамках текущего соединения клиент должен завершить работу протокола Handshake с оповещением unexpected_message(см. 7.2).

Клиент, получивший криптонабор, который не содержится в предложенном им списке криптонаборов, должен завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2). При получении сообщения ClientHello2 сервер должен убедиться в том, что был выбран тот же криптонабор, который указывался сервером в сообщении HelloRetryRequest. После получения сообщения ServerHello клиент должен проверить, что криптонабор, указанный сервером в сообщении ServerHello, совпадает с криптонабором, указанным в сообщении HelloRetryRequest. В противном случае клиент должен завершить работу протокола Handshake с оповещением illegal_parameter(см. 7.2).

В сообщении ClientHello2 клиент не должен предлагать использовать тикеты, которые ассоциируются с хэш-функцией, отличной от хэш-функции, задающейся в рамках выбранного сервером криптонабора, указанного в сообщении HelloRetryRequest.

При получении сообщений ServerHello и HelloRetryRequest клиент должен проверить, что значение поля selected_version расширения supported_versions сообщения ServerHello совпадает с соответствующим значением, указанным в сообщении HelloRetryRequest. В противном случае клиент должен завершить работу протокола Handshake с оповещением illegal_parameter(см. 7.2).

5.6 Расширения

Сообщения протокола TLS 1.3 могут содержать расширения, задающиеся структурой Extension, определяемой следующим образом:

```
struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;
```

где:

- поле extension_type содержит значение типа расширения, которое задается в соответствии с таблицей 2;
- поле extension_data содержит данные, характерные для конкретного расширения.

Настоящие рекомендации определяют набор расширений в соответствии с таблицей 2.

Таблица 2 — Расширения

Название расширения	Тип расширения	Сообщение, в котором оно может пересылаться
server_name	0x0000	CH, EE
supported_groups	0x000A	CH, EE
signature_algorithms	0x000D	CH, CR
pre_shared_key	0x0029	CH, SH
supported_versions	0x002B	CH, SH, HRR
cookie	0x002C	HRR, CH

Окончание таблицы 2

Название расширения	Тип расширения	Сообщение, в котором оно может пересылаться
psk_key_exchange_modes	0x002D	CH
post_handshake_auth	0x0031	CH
signature_algorithms_cert	0x0032	CH, CR
key_share	0x0033	CH, SH, HRR

Примечания

1 В таблице 2 используются следующие обозначения для типов сообщения: CH — ClientHello, SH — ServerHello, EE — EncryptedExtensions, CR — CertificateRequest, HRR — HelloRetryRequest.

2 В настоящих рекомендациях не рассматриваются следующие расширения, указанные в [1]: max_fragment_length(0x0001), status_request(0x0005), use_srtp(0x000E), heartbeat(0x000F), application_layer_protocol_negotiation(0x0010), signed_certificate_timestamp(0x0012), client_certificate_type(0x0013), server_certificate_type(0x0014), padding(0x0015), certificate_authorities(0x002F), oid_filters(0x0030). Настоящие рекомендации не запрещают использовать данные расширения, однако их описание, исследование функционала, предоставляемого данными расширениями, а также анализ стойкости протокола в случае использования данных расширений должны проводиться отдельно.

3 В настоящих рекомендациях расширение early_data, описанное в [1], запрещено к использованию, так как в версии протокола TLS 1.3, соответствующей настоящим рекомендациям, пересылка 0-RTT данных запрещена.

Все реализации должны поддерживать следующие расширения при предоставлении соответствующего функционала:

- расширение supported_versions при согласовании версии протокола TLS 1.3 (см. 5.6.1);
- расширение signature_algorithms при установлении соединения в рамках использования ecdhe_ke схемы аутентифицированной выработки общего ключевого материала для указания поддерживаемых алгоритмов формирования и проверки подписи (см. 5.6.2);
- расширение supported_groups при установлении соединения в рамках использования ecdhe_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала для указания информации о поддерживаемых эллиптических кривых (см. 5.6.3);
- расширение key_share при установлении соединения в рамках использования ecdhe_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала для указания данных, используемых при выработке общего секретного значения *ECDHE* (см. 5.6.4);
- расширения pre_shared_key и psk_key_exchange_modes при установлении соединения в рамках использования psk_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала для указания данных, используемых при согласовании общего секретного значения *PSK* (см. 5.6.5 и 5.6.6);
- расширение post_handshake_auth для поддержки механизма post-handshake аутентификации (см. 5.6.7).

Клиенту рекомендуется поддерживать обработку расширения cookie (см. 5.6.8). Клиенту и серверу рекомендуется поддерживать расширения signature_algorithms_cert и server_name (см. 5.6.2 и 5.6.9).

В большинстве случаев расширения реализованы в формате запрос/ответ, однако в некоторых ситуациях они могут не предполагать ответного расширения. Клиент может послать расширения в качестве запроса в сообщении ClientHello, сервер может послать расширения в качестве ответа в сообщениях HelloRetryRequest, ServerHello, EncryptedExtensions и Certificate. Сервер посылает набор расширений в качестве запроса в сообщениях HelloRetryRequest (для отправки расширения cookie) и CertificateRequest, на которые клиент может ответить с помощью расширений сообщений ClientHello и Certificate соответственно.

Сторона взаимодействия не должна посылать расширения в качестве ответа, если другая сторона не присылала соответствующего расширения в качестве запроса. При получении расширения в качестве ответа на расширение, которое не было послано, сторона взаимодействия должна завершить работу протокола Handshake с оповещением unsupported_extension(см. 7.2).

Если расширение пересылается в сообщении, отличном от определенных в соответствии с таблицей 2 сообщений, сторона взаимодействия, получающая данное расширение, должна завершить работу протокола Handshake с оповещением illegal_parameter(см. 7.2).

В случае присутствия в одном сообщении нескольких различных расширений данные расширения могут быть переданы в произвольном порядке, за исключением расширения pre_shared_key, которое должно быть последним в списке расширений сообщения ClientHello. При этом указанное расширение может появиться в любом месте списка расширений сообщения ServerHello (см. 5.6.5).

5.6.1 Расширение supported_versions

Расширение supported_versions используется для согласования версии протокола TLS, является обязательным в рамках использования любой из схем аутентифицированной выработки общего ключевого материала и посылается клиентом или сервером в следующих случаях:

- клиентом в сообщении ClientHello и содержит список поддерживаемых версий протокола TLS, расположенных в порядке убывания предпочтения. Если клиент поддерживает версию TLS 1.3, расширение supported_versions должно содержать как минимум значение 0x0304. Если поддерживаются предыдущие версии протокола TLS, то они также должны быть указаны в списке поддерживаемых версий;
- сервером в сообщении ServerHello и HelloRetryRequest и содержит информацию о выбранной версии протокола TLS.

Поле extension_data расширения supported_versions задается структурой SupportedVersions, определяемой следующим образом:

```
struct {
    select (Handshake.msg_type) {
        case    client_hello:
                ProtocolVersion versions<2..254>;
        case    server_hello:
                ProtocolVersion selected_version;
    };
} SupportedVersions;
```

где:

- поле versions, указываемое клиентом в структуре SupportedVersions, содержит список поддерживаемых версий протокола TLS, расположенных в порядке убывания предпочтения;
- поле selected_version, указываемое сервером в структуре SupportedVersions, содержит значение версии, выбранной сервером из списка versions структуры SupportedVersions.

Если расширение supported_versions отсутствует в полученном сообщении ClientHello, TLS 1.3 сервер, работающий в режиме совместимости и поддерживающий более ранние версии протокола TLS, должен согласовать версию протокола TLS способом, соответствующим данной версии протокола (например, версию TLS 1.2 в соответствии с Р 1323565.1.020), даже если значение поля ClientHello. legacy_version больше либо равно 0x0304.

Если расширение supported_versions было указано в сообщении ClientHello, сервер не должен использовать значение поля ClientHello.legacy_version для установления версии протокола TLS и должен использовать только информацию, указанную в расширении supported_versions.

Сервер должен выбрать только одну версию протокола TLS из тех, что присутствуют в расширении supported_versions структуры ClientHello, а также игнорировать любые (присутствующие в структуре ClientHello) нераспознанные версии. Сервер должен быть готов получать сообщения ClientHello, содержащие расширение supported_versions, но не содержащие значение 0x0304 в списке поддерживаемых версий.

Сервер, согласовывающий протокол TLS версии ниже 1.3, должен выбрать и зафиксировать значение поля ServerHello.legacy_version и не должен указывать расширение supported_versions в структуре ServerHello. Сервер, согласовывающий протокол TLS версии 1.3, должен указать расширение supported_versions, содержащее выбранное значение версии протокола TLS (то есть 0x0304). При этом сервер должен установить значение поля ServerHello.legacy_version равным 0x0303 (соответствующим значению TLS 1.2). Клиент должен проверить это расширение перед обработкой последующих данных сообщения ServerHello. Если расширение присутствует, клиент должен игнорировать значение поля ServerHello.legacy_version и должен использовать только расширение supported_versions, чтобы определить выбранную сервером версию. Если расширение supported_versions в сообщении ServerHello содержит версию протокола TLS, которая не была предложена клиентом, или версию протокола ниже

протокола TLS 1.3, клиент должен завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

5.6.2 Расширения signature_algorithms и signature_algorithms_cert

Для указания поддерживаемых алгоритмов проверки подписи (и, соответственно, алгоритмов, которые противоположная сторона может использовать для ее формирования) клиент и сервер используют следующие расширения:

- расширение signature_algorithms, содержащее список алгоритмов, которые могут быть использованы для формирования подписи в сообщении CertificateVerify. Данное расширение является обязательным в рамках использования ecdhe_ke схемы аутентифицированной выработки общего ключевого материала и должно быть указано клиентом в сообщении ClientHello и сервером в сообщении CertificateRequest (в случае двусторонней аутентификации). Если аутентификация сервера осуществляется с помощью подписи (в рамках ecdhe_ke схемы), но клиент не указал расширение signature_algorithms, сервер должен завершить работу протокола Handshake с оповещением missing_extension (см. 7.2). Если сообщение CertificateRequest не содержит расширение signature_algorithms, клиент должен завершить работу протокола Handshake с оповещением missing_extension (см. 7.2);
- расширение signature_algorithms_cert, содержащее список алгоритмов подписи, которые могут быть использованы для формирования подписей сертификатов из цепочки сертификатов соответствующей стороны. Данное расширение является опциональным и посылается в рамках использования ecdhe_ke схемы аутентифицированной выработки общего ключевого материала и может быть указано клиентом в сообщении ClientHello и сервером в сообщении CertificateRequest (в случае двусторонней аутентификации). Если данное расширение не указывается стороной взаимодействия, считается, что допустимые для подписи сертификатов алгоритмы задаются расширением signature_algorithms.

Открытые ключи, присутствующие в сертификатах сообщения Certificate, должны соответствовать согласованным в рамках обмена расширениями signature_algorithms и signature_algorithms_cert алгоритмам подписи.

Поле extension_data в расширениях signature_algorithms_cert и signature_algorithms задается структурой SignatureSchemeList, определяемой следующим образом:

```
struct {
    SignatureScheme supported_signature_algorithms<2..2^16-2>;
} SignatureSchemeList;
```

где поле supported_signature_algorithms содержит список схем подписи, каждая из которых задает алгоритм подписи и используемую эллиптическую кривую. Значения элементов списка supported_signature_algorithms, допустимые к использованию в рамках данного документа, задаются в 10.2.

Элементы поля supported_signature_algorithms указываются в порядке убывания предпочтения.

Примечание — Значение подписи под корневым сертификатом обычно не проверяется при проверке цепочки сертификатов, указанной в сообщении Certificate. Поэтому алгоритм подписи, с помощью которого подписан этот сертификат, может не удовлетворять значениям, указанным в расширениях signature_algorithms и signature_algorithms_cert.

5.6.3 Расширение supported_groups

Расширение supported_groups используется для передачи информации о поддерживаемых сторонами взаимодействия эллиптических кривых и посылается клиентом или сервером в следующих случаях:

- клиентом в сообщении ClientHello и содержит необходимую для выработки общего энтропийного значения *ECDHE* информацию о кривых, поддерживаемых клиентом. Данное расширение является обязательным в рамках использования ecdhe_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала;
- сервером в сообщении EncryptedExtensions и содержит информацию о кривых, поддерживаемых сервером, не влияющую на выработку общего энтропийного значения *ECDHE* в текущем соединении и предназначенную для информирования клиента. Данное расширение является опциональным и посылается в рамках использования ecdhe_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала.

Поле extension_data расширения supported_groups задается структурой NamedGroupList, определяемой следующим образом:

```
struct {
    NamedGroup named_group_list<2..2^16-1>;
} NamedGroupList;
```

где поле named_group_list содержит список кривых, поддерживаемых стороной взаимодействия, в порядке убывания предпочтения. Значения элементов списка named_group_list, допустимые к использованию в рамках данного документа, задаются в 10.3.

Если у сервера есть кривая, которая является предпочтительнее кривых, указанных клиентом в расширении key_share, и при этом сервер желает принять сообщение ClientHello, ему следует послать расширение supported_groups, чтобы предоставить клиенту информацию о своих предпочтениях относительно кривых. В этом расширении серверу рекомендуется указать все поддерживаемые им кривые, вне зависимости от того, поддерживаются ли они клиентом. Клиент не должен реагировать на информацию, полученную от сервера в расширении supported_groups, до момента успешного выполнения протокола Handshake, но может использовать полученную информацию для того, чтобы изменить список кривых, используемых им, в расширении key_share в последующих соединениях.

В случае получения от клиента расширения supported_groups, не содержащего поддерживаемые сервером кривые, сервер должен завершить работу протокола Handshake либо с оповещением handshake_failure, либо с оповещением insufficient_security (см. 7.2).

5.6.4 Расширение key_share

Расширение key_share отвечает за выработку общего энтропийного значения *ECDHE*, является обязательным в рамках использования ecdhe_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала и посылается клиентом или сервером в следующих случаях:

- клиентом в сообщении ClientHello (см. подробнее 5.6.4.1) и содержит информацию об открытых эфемерных ключах Q^1_c, … , Q^N_c, $N \geq 1$, предлагаемых клиентом;
- сервером в сообщении HelloRetryRequest (см. подробнее 5.6.4.2) и содержит идентификатор кривой, выбранной сервером из списка named_group_list сообщения ClientHello (см. 5.6.3);
- сервером в сообщении ServerHello (см. подробнее 5.6.4.3) и содержит информацию об открытом эфемерном ключе Q_s, принадлежащем кривой, которой соответствует открытый эфемерный ключ клиента Q_c из списка client_shares.

При этом информация о каждом открытом эфемерном ключе задается структурой KeyShareEntry следующим образом:

```
struct {
    NamedGroup group;
    opaque key_exchange<1..2^16-1>;
} KeyShareEntry;
```

где group — поле длиной в 2 байта, в котором указывается идентификатор кривой;

key_exchange — открытый эфемерный ключ, принадлежащий кривой, идентификатор которой указан в поле group. Значение данного поля определяется в соответствии с 8.5 и представляется в формате, указанном в 5.6.4.4.

5.6.4.1 Расширение key_share в сообщении ClientHello

В сообщении ClientHello поле extension_data расширения key_share задается структурой KeyShareClientHello, определяемой следующим образом:

```
struct {
    KeyShareEntry client_shares<0..2^16-1>;
} KeyShareClientHello;
```

где поле client_shares содержит список предлагаемых клиентом структур KeyShareEntry, перечисленных в порядке убывания предпочтения. Данный список может быть пустым (таким образом, клиент может запросить от сервера сообщение HelloRetryRequest, чтобы вычислять значение открытого эфемерного

ключа только для кривой, поддерживаемой сервером). Каждый элемент поля client_shares должен соответствовать кривой, указанной в поле named_group_list расширения supported_groups. Порядок элементов поля client_shares должен соответствовать порядку кривых, указанных в поле named_group_list расширения supported_groups. При этом в поле client_shares могут отсутствовать элементы, соответствующие некоторым (в том числе наиболее предпочтительным для сервера) кривым, указанным в поле named_group_list.

Примечание — Подобная ситуация может возникнуть, если наиболее предпочтительные группы являются новыми и вероятно не будут поддерживаться в достаточном числе реализаций.

Количество элементов поля client_shares не должно превышать количество кривых, указанных в поле named_group_list расширения supported_groups. При этом каждой кривой соответствует только один элемент поля client_shares, и клиент не должен предлагать большее количество структур для одной и той же группы. Значения полей key_exchange каждой структуры KeyShareEntry должны быть сформированы независимым образом. Клиент не должен предлагать элементы поля client_shares для кривых, не указанных в поле named_group_list расширения supported_groups. В случае нарушений указанных выше правил серверу рекомендуется завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

5.6.4.2 Расширение key_share в сообщении HelloRetryRequest

В сообщении HelloRetryRequest поле extension_data расширения key_share задается структурой KeyShareHelloRetryRequest, определяемой следующим образом:

```
struct {
    NamedGroup selected_group;
} KeyShareHelloRetryRequest;
```

где поле selected_group содержит кривую, поддерживаемую обеими сторонами, которую сервер предпочитает согласовать.

Если сервер указал расширение key_share в сообщении HelloRetryRequest, клиент должен проверить, что:

- поле selected_group соответствует одной из кривых, указанных в поле named_group_list расширения supported_groups сообщения ClientHello1;
- поле selected_group не соответствует никакой кривой, указанной в расширении key_share сообщения ClientHello1.

Если не выполняется любое из перечисленных условий, клиент должен завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2). В противном случае при отправлении сообщения ClientHello2 клиент должен заменить список в поле client_shares расширения key_share на единственный элемент, в котором в поле group указана кривая, соответствующая кривой из поля selected_group сообщения HelloRetryRequest, а в поле key_exchange содержится открытый эфемерный ключ, принадлежащий кривой, указанной в поле group.

5.6.4.3 Расширение key_share в сообщении ServerHello

В сообщении ServerHellonоле extension_data расширения key_share задается структурой KeyShareServerHello, определяемой следующим образом:

```
struct {
    KeyShareEntry server_share;
} KeyShareServerHello;
```

где поле server_share имеет структуру KeyShareEntry, в которой поле group должно соответствовать некоторой кривой, для которой указан элемент в поле client_shares.

В рамках ecdhe_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала сервер указывает ровно одну структуру KeyShareEntry в сообщении ServerHello. Сервер не должен посылать структуру KeyShareEntry для кривой, не указанной клиентом в расширении supported_groups. Сервер не должен посылать расширение key_share в рамках psk_ke схемы аутентифициро-

ванной выработки общего ключевого материала. В случае если клиентом было получено сообщение HelloRetryRequest с расширением key_share, при получении сообщения ServerHello клиент должен убедиться, что идентификатор кривой, указанный в сообщении ServerHello, совпадает с идентификатором кривой, указанным в сообщении HelloRetryRequest. Если указанное условие не выполняется, клиент должен завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

5.6.4.4 Представление открытых эфемерных ключей

Каждый открытый эфемерный ключ Q, записываемый в поле key_exchange структуры KeyShareEntry, задается следующим образом:

```
struct {
    opaque X[coordinate_length];
    opaque Y[coordinate_length];
} PlainPointRepresentation;
```

где:

- поля X и Y содержат байтовые представления координат x и y точки $Q(Q = (x, y))$ в формате little-endian, сформированные следующим образом:

$$X = str_{coordinate_length}(x),$$
$$Y = str_{coordinate_length}(y); \quad (1)$$

- значение параметра coordinate_length определяется в соответствии с таблицей 17.

Примечание — В соответствии с [1] допускается использование других форматов представления открытого эфемерного ключа, однако при работе протокола в соответствии с настоящими рекомендациями данный функционал запрещен.

5.6.5 Расширение pre_shared_key

Расширение pre_shared_key отвечает за согласование общего энтропийного значения *PSK*, является обязательным в рамках использования psk_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала и посылается клиентом или сервером в следующих случаях:

- клиентом в сообщении ClientHello и содержит список данных, соответствующих предлагаемым PSK-значениям, и последовательность связанных с ними binder-значений (см. 5.6.5.3);
- сервером в сообщении ServerHello и содержит индекс элемента, выбранного сервером из полученного от клиента списка данных, соответствующих PSK-значениям.

Поле extension_data расширения pre_shared_key содержит элемент PreSharedKeyExtension, задающийся следующей структурой:

```
struct {
    select (Handshake.msg_type) {
        case client_hello:   OfferedPsks;
        case server_hello:  uint16 selected_identity;
    };
} PreSharedKeyExtension;
```

где структура OfferedPsks и индекс selected_identity определены в 5.6.5.1 и 5.6.5.2 соответственно.

5.6.5.1 Структура OfferedPsks

Структура OfferedPsks задается следующим образом:

```
opaque PskBinderEntry<32..255>;
struct {
    PskIdentity identities<7..2^16-1>;
    PskBinderEntrybinders<33..2^16-1>;
} OfferedPsks;
```

где identities — список данных, соответствующих значениям *PSK*, предлагаемым клиентом для согласования общего энтропийного значения;

binders — последовательность binder-значений, соответствующих каждой структуре PskIdentity из списка identities и указанных в том же порядке. Процесс вычисления binder-значений описан в 5.6.5.3.

Структура PskIdentity, содержащая данные, соответствующие значению *PSK*, задается следующим образом:

```
struct {
    opaque identity<1..2^16-1>;
    uint32 obfuscated_ticket_age;
} PskIdentity;
```

где identity — идентификатор (тикет) значения *PSK*. Для значения *iPSK* данное поле принимает значение поля NewSessionTicket.ticket (см. 5.9.1.1). Настоящие рекомендации не фиксируют механизм формирования тикета для значения *ePSK* (см. 5.2);

obfuscated_ticket_age — маскированное время жизни тикета.

Для каждого значения *iPSK* значение данного поля равно времени жизни (clients_ticket_age) тикета на стороне клиента, сложенному по модулю 2^{32} со значением поля NewSessionTicket.ticket_age_add. При этом время жизни тикета на стороне клиента исчисляется в миллисекундах, прошедших с момента получения соответствующего сообщения NewSessionTicket со стороны сервера.

Клиент не должен использовать значение *iPSK*, время жизни тикета которого превышает указанное в поле NewSessionTicket.ticket_lifetime время жизни тикета, ассоциированного с данным значением *iPSK*.

Для каждого значения *ePSK* значение данного поля рекомендуется устанавливать равным 0, при этом серверы должны игнорировать данное значение.

5.6.5.2 Значение selected_identity

Значение selected_identity, указываемое сервером в структуре PreSharedKeyExtension, определяется как индекс элемента, выбранного сервером из списка identities структуры OfferedPsks. При этом индексация элементов в указанном списке начинается с нуля.

С каждым тикетом, соответствующим значению *PSK*, однозначно ассоциируется алгоритм хэширования *HASH* (см. подробнее 5.9.1.1). При этом алгоритм хэширования для значения *iPSK* должен соответствовать алгоритму, который использовался в инициализирующем соединении. Сервер должен убедиться в том, что он выбирает криптонабор и тикет, соответствующие одинаковому алгоритму хэширования.

В рамках работы iPSK-only и iPSK-ECDHE режимов наиболее простым способом реализации соответствия значения тикета криптонабору является согласование криптонабора и затем исключение несовместимых с ним тикетов. Любые нераспознанные значения тикетов (то есть те, которые не были найдены в базе данных тикетов или были зашифрованы на нераспознанном ключе) должны быть проигнорированы. В случае если сервер не обнаружил ни одного тикета, соответствующего выбранному им криптонабору, он должен по возможности продолжить работу протокола Handshake в ECDHE-only режиме, в противном случае сервер должен завершить работу протокола Handshake с оповещением unknown_psk_identity.

Прежде чем подтвердить выбор тикета (и соответствующего ему значения *PSK*), сервер должен выполнить следующие проверки:

- необходимо проверить, что binder-значение (см. 5.6.5.3), соответствующее данному значению *PSK*,сформировано корректно. Если данное значение отсутствует или сформировано некорректно, сервер должен завершить работу протокола Handshake с оповещением decrypt_error (см. 7.2). Сервер не должен проверять все binder-значения, вместо этого он должен выбрать только один тикет (*PSK*) и проверить binder-значение, соответствующее выбранному *PSK*. В качестве подтверждения выбора тикета сервер указывает значение selected_identity в расширении pre_shared_key;

- в случае если выбранное значение selected_identity соответствует значению *iPSK*, необходимо убедиться, что время жизни данного тикета на стороне сервера не превышает максимально допустимого значения (см. 5.9.1);

- в случае если выбранное значение selected_identity соответствует значению *iPSK*, сервер должен извлечь из полученного от клиента значения obfuscated_ticket_age значение времени жизни тикета на стороне клиента clients_ticket_age, используя значение ticket_age_add (данное значение должно быть известно серверу, см. подробнее 5.9.1), и сравнить его с ожидаемым значением expected_ticket_age, полученным путем вычитания времени создания тикета creation_time (значение которого должно быть известно серверу, см. подробнее 5.9.1) из текущего времени: разница между значениями clients_ticket_age и expected_ticket_age не должна превышать допустимой задержки в канале связи, установленной политикой безопасности сервера.

Если хотя бы одна из указанных выше проверок не прошла успешно, сервер должен завершить работу протокола Handshake с оповещением handshake_failure (см. 7.2).

Получив расширение pre_shared_key от сервера, клиент должен проверить, что:

- выбранное сервером значение selected_identity находится в пределах длины списка identities структуры OfferedPsks, предоставленной клиентом;

- сервер выбрал криптонабор, включающий алгоритм хэширования, который ассоциируется со значением тикета, соответствующим значению selected_identity;

- в рамках psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала в сообщении ServerHello указано расширение key_share.

Если не выполняется хотя бы одно из указанных выше требований, клиент должен завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

Расширение pre_shared_key должно быть указано последним в списке extensions сообщения ClientHello. Сервер должен убедиться, что данное расширение указано последним, и в противном случае завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

5.6.5.3 Binder-значение

Binder-значение содержит контрольные данные, сформированные с помощью функции *HMAC* и предназначенные для проверки корректности значения *PSK*, соответствующего данному binder-значению, на стороне сервера. Binder-значение, соответствующее *iPSK*, связывает инициализирующее соединение с текущим соединением.

Список всех binder-значений, предлагаемых клиентом, указывается в поле binders расширения pre_shared_key. Каждое binder-значение имеет тип PskBinderEntry, описанный в 5.6.5.1, и вычисляется следующим образом:

$$\begin{aligned} binder &= HMAC(HMAC_binder_key,\ Transcript\text{-}Hash\ (Messages)) \\ HMAC_binder_key &= HKDF\text{-}Expand\text{-}Label\ (BS,\ "finished",\ "",\ HLen), \end{aligned} \qquad (2)$$

где *BS* (*binder_secret*) — секретное значение, определенное в 8.2;

HKDF-Expand-Label — функция, определенная в 8.1.3;

Transcript-Hash — хэш-функция, определенная в 8.8;

Messages — упорядоченное множество строк, являющихся входными параметрами хэш-функции *Transcript-Hash*, задающееся следующим образом.

Если сервером не было послано сообщение HelloRetryRequest, то:

$$Messages = \{Truncate(ClientHello)\}. \qquad (3)$$

Если сервером было послано сообщение HelloRetryRequest, то:

$$Messages = \{ClientHello1,\ HelloRetryRequest,\ Truncate\ (ClientHello2\ )\}. \qquad (4)$$

Здесь *Truncate*(*ClientHello* — строка, соответствующая сообщению ClientHello, в котором удалено поле binders структуры OfferedPsks расширения pre_shared_key. При этом длины сообщения ClientHello, поля extensions и расширения pre_shared_key не изменяются (остаются такими же, как если бы поле binders не исключалось).

Примечание — Поле binders удаляется из сообщения ClientHello для того, чтобы binder-значение, соответствующее одному значению *PSK* из списка, предложенного клиентом, не зависело от остальных binder-значений, соответствующих значениям *PSK* из данного списка.

5.6.6 Расширение psk_key_exchange_modes

Расширение psk_key_exchange_modes отвечает за согласование режима использования значения *PSK*, является обязательным в рамках использования psk_ke и psk_ecdhe_ke схем аутентифицированной выработки общего ключевого материала и посылается только клиентом в сообщении ClientHello.

Данное расширение содержит информацию о режимах использования PSK-значений, которые клиент готов поддержать. Если клиент указывает расширение pre_shared_key без указания расширения psk_key_exchange_modes, сервер должен завершить выполнение протокола Handshake с оповещением missing_extension (см. 7.2). Данное расширение также налагает требования по использованию PSK-значений, соответствующих тикетам (см. 5.9.1), посланным в сообщениях NewSessionTicket в рамках текущего соединения: значения *iPSK*, соответствующие данным тикетам, не должны использоваться в последующих соединениях в рамках режимов, не поддерживаемых в текущем соединении.

Сервер не должен выбирать режим работы, не указанный в расширении psk_key_exchange_modes.

Поле extension_data расширения psk_key_exchange_modes задается структурой PskKeyExchangeModes, определяемой следующим образом:

```
enum {
    psk_ke(0x00),
    psk_dhe_ke(0x01),
    (0xFF)
} PskKeyExchangeMode;
struct {
    PskKeyExchangeMode ke_modes<1..255>;
} PskKeyExchangeModes;
```

где поле ke_modes содержит список следующих значений, предлагаемых клиентом:

- значение psk_ke(0x00), соответствующее режиму использования предлагаемых PSK-значений в рамках psk_ke схемы аутентифицированной выработки общего ключа;
- значение psk_dhe_ke(0x01), соответствующее режиму использования предлагаемых PSK-значений в рамках psk_ecdhe_ke схемы аутентифицированной выработки общего ключа.

5.6.7 Расширение post_handshake_auth

Расширение post_handshake_auth используется для указания того, что клиент поддерживает возможность проведения post-handshake аутентификации (см. подробнее 5.9.2), является опциональным в рамках использования любой из схем аутентифицированной выработки общего ключевого материала и посылается только клиентом в сообщении ClientHello.

Если клиент поддерживает процедуру post-handshake аутентификации, он должен указать расширение post_handshake_auth в сообщении ClientHello. При получении данного расширения от клиента сервер может отправить запрос на аутентификацию клиента путем пересылки сообщения CertificateRequest (см. 5.7.2) в любой момент времени после получения main-handshake сообщения Finished со стороны клиента.

Если данное расширение не было указано в сообщении ClientHello, то сервер не должен запрашивать аутентификацию клиента с помощью post-handshake сообщения CertificateRequest, в противном случае клиент должен ответить оповещением об ошибке unexpected_message (см. 7.2).

Поле extension_data расширения post_handshake_auth содержит пустой вектор и определяется следующей структурой:

```
struct {} PostHandshakeAuth;
```

5.6.8 Расширение cookie

Расширение cookie используется для передачи сервером некоторых данных для хранения на стороне клиента. Данное расширение является опциональным в рамках использования любой из схем аутентифицированной выработки общего ключевого материала и может быть послано клиентом и сервером в следующих случаях:

- сервером в сообщении HelloRetryRequest, что является исключением из правила, согласно которому сервер может отправлять только те расширения, которые присутствуют в сообщении ClientHello;

Примечание — Данное расширение может содержать хэш значение, сформированное от сообщения ClientHello1, что позволит серверу не хранить сообщение ClientHello1 (см. 8.8) и отличать сообщения ClientHello1 и ClientHello2.

- клиентом в сообщении ClientHello2 в качестве ответа на сообщение HelloRetryRequest, содержащее данное расширение.

Поле extension_data расширения cookie задается структурой Cookie, определяемой следующим образом:

```
struct {
    opaque cookie<1..2^16-1>;
} Cookie;
```

Настоящие рекомендации не фиксируют механизм формирования содержимого данного расширения на стороне сервера. В случае использования данного расширения его описание, исследование функционала, предоставляемого данным расширением, а также анализ стойкости протокола должны проводиться отдельно.

В целях совместимости различных реализаций клиентам рекомендуется всегда поддерживать обработку данного расширения посредством пересылки содержимого расширения cookie, полученного в сообщении HelloRetryRequest, в ответном сообщении ClientHello2 без каких-либо дополнительных изменений.

Клиенты не должны указывать расширение cookie в сообщениях ClientHello, посылаемых не в ответ на сообщение HelloRetryRequest.

5.6.9 Расширение server_name

В случае если на сервере с некоторым IP адресом размещается несколько хостов со своим уникальным именем и сертификатом (не обязательно уникальным), предоставляющих некоторый сервис, будем называть данные хосты виртуальными серверами с одним IP адресом.

Расширение server_name используется для выбора виртуального сервера, с которым клиент устанавливает соединение, является опциональным в рамках использования любой из схем аутентифицированной выработки общего ключевого материала и посылается клиентом или сервером в следующих случаях:

- клиентом в сообщении ClientHello для указания имени виртуального сервера, с которым клиент устанавливает соединение. Данный механизм предоставляет возможность различным виртуальным серверам на одном IP-адресе использовать различные сертификаты;
- сервером в сообщении EncryptedExtensions для информирования о том, что запрос клиента по выбору соответствующего сервиса был учтен.

Поле extension_data расширения server_name формируется следующим образом:

- в случае, если данное расширение посылается сервером, поле должно содержать пустой вектор;
- в случае, если данное расширение посылается клиентом, поле должно содержать структуру ServerNameList, определяемую следующим образом:

```
struct {
    ServerName server_name_list<1..2^16-1>
} ServerNameList;

struct {
    NameType name_type;
    select (name_type) {
        case host_name: HostName;
    } name;
} ServerName;
```

```
enum {
    host_name(0), (255)
} NameType;
opaqueHostName<1..2^16-1>;
```

Примечание — В соответствии с [1] допускается возможность использования других типов имен, отличных от указанных в перечислении NameType. Настоящие рекомендации не запрещают использовать данные типы, однако их описание, исследование функционала, а также анализ стойкости протокола в случае использования данных типов имен должны проводиться отдельно.

Поле server_name_list не должно содержать более одного имени, соответствующего типу NameType. В случае если сервер не может распознать имя, указанное клиентом в данном расширении, он может либо завершить работу протокола Handshake с оповещением unrecognized_name (см. 7.2), либо продолжить работу протокола Handshake.

Поле HostName содержит полное доменное имя виртуального сервера, представляющее собой байтовую строку в кодировке ASCII без точки в конце своей записи. Указанное представление позволяет поддерживать использование многоязычных доменных имен. Поле HostName не должно содержать IPv4 или IPv6 адрес в качестве своего значения.

Если клиент возобновляет соединение на основе значения *iPSK*, которое было выработано в соответствии с 8.6 в результате цепочки соединений, где первоначальное соединение устанавливалось в рамках ecdhe_ke схемы аутентифицированной выработки общего ключевого материала, и отправляет расширение server_name, то он должен указать в данном расширении значение, являющееся действительным для сертификата виртуального сервера, отправленного в рамках первоначального соединения. При этом клиенту рекомендуется указывать то же значение, которое было указано им в расширении server_name в рамках первоначального соединения для выбора виртуального сервера. Данная рекомендация предназначена для оптимизации производительности, поскольку часто различные виртуальные серверы, использующие один и тот же сертификат, могут не поддерживать механизм возобновления соединения на основе значения *PSK*, выработанного в рамках соединения с другим виртуальным сервером.

5.7 Параметры сервера

5.7.1 Сообщение EncryptedExtensions

Данное сообщение является обязательным, отправляется сервером сразу после сообщения ServerHello во всех режимах работы протокола Handshake и является первым сообщением, передающимся в защищенном виде с помощью ключей, выработанных на основе секретного значения [*sender*]_*handshake_traffic_secret* (см. 8.2). Данное сообщение не посылается клиентом.

Сообщение EncryptedExtensions содержит список расширений extensions, содержащих информацию, не влияющую на выработку общего ключевого материала и не ассоциированную с данными о конкретных сертификатах. Список всех расширений, которые могут посылаться в рамках сообщения EncryptedExtensions, указан в таблице 2.

Клиент должен проверить наличие запрещенных расширений в сообщении EncryptedExtensions и, в случае если они есть, завершить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

Структура EncryptedExtensions сообщения EncryptedExtensions задается следующим образом:

```
struct {
    Extension extensions<0..2^16-1>;
} EncryptedExtensions;
```

5.7.2 Сообщение CertificateRequest

Данное сообщение является опциональным и посылается сервером в случае необходимости запроса на аутентификацию клиента на основе сертификатов и подписи.

Сервер может отправить сообщение CertificateRequest в одном из следующих случаев:

- во время работы протокола Handshake в рамках ecdhe_ke схемы аутентифицированной выработки общего ключевого материала при обмене main-handshake сообщениями, причем данное сообщение должно следовать за сообщением EncryptedExtensions;

- в рамках post-handshake аутентификации при условии, что ранее клиент отправлял расширение post_handshake_auth в сообщении ClientHello (см. 5.9.2).

В случае если режим работы протокола Handshake подразумевает использование предварительно распределенного секрета (в рамках psk_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала), сообщение CertificateRequest не должно посылаться сервером в рамках обмена main-handshake сообщениями. Клиент, получивший сообщение CertificateRequest в указанном случае, должен завершить работу протокола Handshake с оповещением unexpected_message (см. 7.2).

Структура CertificateRequest сообщения CertificateRequest задается следующим образом:

```
struct {
    opaque certificate_request_context<0..2^8-1>;
    Extension extensions<2..2^16-1>;
} CertificateRequest;
```

где certificate_request_context — идентификатор сообщения CertificateRequest.

Если запрос посылается сервером при обмене main-handshake сообщениями, данное поле содержит вектор нулевой длины.

Если запрос посылается сервером в рамках post-handshake аутентификации (см. 5.9.2), данное поле должно содержать уникальное значение в рамках текущего соединения. Данное значение также рекомендуется выбирать непредсказуемым образом в целях противодействия атакам, в рамках которых противник имеет возможность получать кратковременный доступ к интерфейсу подписания произвольных сообщений с помощью ключа подписи клиента;

extensions — набор расширений, задающих параметры запрашиваемого сертификата. Расширение signature_algorithms, описанное в 5.6.2, является обязательным. Опционально может быть указано расширение signature_algorithms_cert. Настоящие рекомендации не запрещают использовать другие расширения, разрешенные для данного сообщения, однако их описание, исследование функционала, предоставляемого данными расширениями, а также анализ стойкости протокола в случае использования данных расширений должны проводиться отдельно. Нераспознанные расширения должны игнорироваться клиентом.

5.8 Сообщения аутентификации

5.8.1 Сообщение Certificate

Данное сообщение может посылаться сервером или клиентом и содержит цепочку сертификатов отправителя.

Сервер должен отправлять сообщение Certificate только при использовании схемы ecdhe_ke аутентифицированной выработки общего ключевого материала.

Клиент должен отправлять сообщение Certificate в том и только в том случае, если сервер послал ему сообщение CertificateRequest (см. 5.7.2).

Структура Certificate сообщения Certificate задается следующим образом:

```
struct {
    opaque cert_data<1..2^24-1>;
    Extension extensions<0..2^16-1>;
} CertificateEntry;

struct {
    opaque certificate_request_context<0..2^8-1>;
    CertificateEntry certificate_list<0..2^24-1>;
} Certificate;
```

где certificate_request_context — идентификатор запроса сертификата. Если сообщение Certificate посылается сервером, то данное поле должно оставаться пустым.

Если данное сообщение посылается клиентом, то поле certificate_request_context должно содержать то же значение, что и поле certificate_request_context сообщения CertificateRequest;

certificate_list — цепочка сертификатов, представленная в виде последовательности структур CertificateEntry, каждая из которых содержит один сертификат в формате x.509[1)], описанном в Р 1323565.1.023, в кодировке DER и набор опциональных расширений. Расширения, пересылаемые в этом сообщении сервером, должны соответствовать расширениям, посланным ему клиентом в сообщении ClientHello. Расширения, посылаемые в сообщении Certificate клиентом, должны соответствовать расширениям, которые были ранее указаны сервером в сообщении CerificateRequest.

Сертификат отправителя должен следовать первым в данной цепочке и содержать ключ проверки подписи Q_{verify}, посылаемой отправителем в сообщении CertificateVerify.

Сертификаты в цепочке рекомендуется располагать в таком порядке, чтобы каждый следующий сертификат подтверждал подлинность ключа проверки подписи предыдущего. В некоторых случаях корневые сертификаты могут не добавляться в пересылаемую цепочку (например, если известно, что сторона-получатель обладает этим корневым сертификатом).

В сообщении Certificate, посылаемом сервером, поле certificate_list не должно быть пустым. Если сервер посылает сообщение Certificate с пустым полем certificate_list, клиент должен завершить работу протокола Handshake с оповещением decode_error (см. 7.2).

Примечание — Если сервер не может сформировать цепочку сертификатов, подписанных с помощью алгоритмов, задающихся с помощью схем подписи, указанных клиентом в расширении signature_algorithms_cert (signature_algorithms) (например, вследствие того, что сертификаты в цепочке сертификатов сервера подписаны устаревшими алгоритмами подписи, для которых не существует идентификаторов, которые клиент может указать в расширении signature_algorithms_cert), он может попробовать указать цепочку сертификатов по своему усмотрению. Клиент, получив сообщение Certificate, содержащее сертификат, подписанный алгоритмом, не поддерживаемым клиентом, должен завершить соединение с оповещением bad_certificate (см. 7.2).

Если сервер послал сообщение CertificateRequest, клиент должен послать серверу сообщение Certificate. Если у клиента нет сертификата, соответствующего параметрам, указанным сервером в сообщении CertificateRequest (или если клиент решает отклонить запрос об аутентификации, см. 5.9.2), то клиент должен указать поле certificate_list пустым (нулевой длины).

Если клиент не может сформировать цепочку сертификатов, подписанных с помощью поддерживаемых сервером алгоритмов (алгоритмы задаются в соответствии с 5.6.2), и при этом решает завершить работу протокола Handshake, то он должен завершить работу протокола Handshake с соответствующим оповещением (по умолчанию таким является unsupported_certificate, см. 7.2).

Если клиент указал поле certificate_list пустым в сообщении Certificate, сервер может по своему усмотрению либо продолжить работу протокола Handshake без аутентификации клиента, либо завершить работу протокола Handshake с оповещением certificate_required. Кроме того, если какой-то из сертификатов в цепочке сертификатов клиента оказался неподдерживаемым (например, не был издан доверенным удостоверяющим центром), сервер может либо продолжить работу протокола Handshake (считая при этом, что клиент не аутентифицировался), либо завершить работу протокола Handshake с оповещением bad_certificate (см. 7.2).

Примечание — Следует отметить, что сертификат, содержащий ключ, соответствующий одному алгоритму подписи, может быть подписан с помощью другого алгоритма подписи, однако все используемые алгорит-

[1)] В соответствии с [1] протокол TLS 1.3 допускает возможность использования сертификатов формата RawPublicKey, содержащих только открытый ключ, однако настоящие рекомендации не описывают данный вариант использования сертификатов. При необходимости использования сертификатов в формате RawPublicKey описание данного формата, исследование функционала, предоставляемого данным форматом, а также анализ стойкости протокола в случае использования данного формата должны проводиться отдельно.

мы подписи должны соответствовать схемам подписи, которые вторая сторона указала в расширении signature_algorithms_cert (или расширении signature_algorithms в случае если расширение signature_algorithms_cert не было указано). При этом для криптонаборов, описанных в настоящих рекомендациях (см. 10.1), могут использоваться только те алгоритмы подписи, которые соответствуют схемам подписи, перечисленным в 10.2.

5.8.2 Сообщение CertificateVerify

За счет пересылки данного сообщения осуществляется доказательство того, что отправитель обладает ключом подписи, соответствующим ключу проверки подписи, переданному им ранее в сообщении Certificate. Также с помощью данного сообщения обеспечивается целостность переданных ранее сообщений протокола Handshake.

Если сторона посылает сообщение CertificateVerify, то оно посылается непосредственно после сообщения Certificate и перед сообщением Finished.

Сервер должен отправлять сообщение CertificateVerify только при использовании схемы ecdhe_ke аутентифицированной выработки общего ключевого материала. Поскольку сервер не может отправить пустой список сертификатов в сообщении Certificate, то сервер должен посылать сообщение CertificateVerify всегда.

Клиент должен отправить сообщение CertificateVerify, если сервер запросил аутентификацию клиента с помощью сообщения CertificateRequest и поле certificate_list, указанное клиентом в пересланном ранее сообщении Certificate, не являлось пустым.

Данное сообщение не должно пересылаться клиентом, в случае если он указал пустой список сертификатов в сообщении Certificate.

Структура CertificateVerify сообщения CertificateVerify задается следующим образом:

```
struct {
    SignatureScheme algorithm;
    opaque signature<0..2^16-1>;
} CertificateVerify;
```

где algorithm — значение схемы подписи, задающей алгоритм подписи и используемую эллиптическую кривую. Значения, допустимые к использованию в рамках данного документа, задаются в 10.2 в перечислении SignatureScheme.

Алгоритм подписи, задающийся схемой подписи, должен соответствовать алгоритму проверки подписи, указанному в сертификате соответствующей стороны в сообщении Certificate.

Если сервер отправляет сообщение CertificateVerify, значение схемы подписи должно соответствовать одному из указанных значений схем подписи в списке supported_signature_algorithms расширения signature_algorithms сообщения ClientHello. Исключением является случай, когда сервер не может сформировать цепочку сертификатов, подписанных с помощью указываемых клиентом схем подписи (см. 5.8.1).

Если клиент отправляет сообщение CertificateVerify, значение схемы подписи должно соответствовать одному из указанных значений схем подписи в списке supported_signature_algorithms расширения signature_algorithms сообщения CertificateRequest;

signature — значение подписи, формируемое стороной с помощью схемы подписи, значение которой указано в поле algorithm.

Значение подписи *sgn*, указываемой стороной в поле signature, формируется следующим образом:

$$sgn = SIGN(c|context|0x00|HM, d_sign), \quad (5)$$

где *SIGN* — функция формирования подписи, задаваемая схемой подписи (см. 10.2);

c — строка, содержащая 64 подряд идущих байта 0x20;

context — строковая константа, принимающая следующее значение: *"TLS* 1.3, *server CertifcateVerify"* в случае формирования подписи на стороне сервера, *"TLS* 1.3, *client CertifcateVerify"* в случае формирования подписи на стороне клиента;

HM — байтовая строка, соответствующая значению *Transcript-Hash* (*Handshake Context, Certificate*), где параметр *Handshake Context* определяется в соответствии с 8.9.

Отправитель подписывает вышеперечисленные данные с помощью ключа подписи d_{sign}, которому однозначно соответствует ключ проверки подписи Q_{verify}, указанный в сертификате отправителя в сообщении Certificate.

Получатель проверяет подпись, указанную в поле signature сообщения CertificateVerify, с помощью ключа проверки подписи Q_{verify}, полученного из сертификата отправителя, переданного им ранее в сообщении Certificate. Если проверка подписи не завершилась успешно, получатель должен завершить работу протокола Handshake с оповещением decrypt_error(см. 7.2).

5.8.3 Сообщение Finished

Сообщение Finished является последним сообщением, пересылаемым в рамках сообщений аутентификации протокола Handshake, и посылается в следующих случаях:

- сервером или клиентом в рамках обмена main-handshake сообщениями;
- клиентом в рамках post-handshake аутентификации (см. 5.9.2).

Структура Finished сообщения Finished задается следующим образом:

```
struct {
    opaque verify_data[Hash.length];
} Finished;
```

где Hash.length принимает значение *HLen* (см. подробнее 10.1.4), а значение поля verify_data вычисляется следующим образом:

$$verify_data = HMAC([sender]_finished_key, Transcript\text{-}Hash(Handshake\ Context, Certificate^*, CertificateVerify^*)), \quad (6)$$

где

- параметр *Handshake Context* определяется в соответствии с 8.9;
- параметры *Certificate**, *Certificate Verify** являются строками, соответствующими байтовым представлениям сообщений Certificate, CertificateVerify, если соответствующие сообщения были отправлены в рамках протокола Handshake, и пустыми строками в противном случае;
- ключ вычисления кода аутентификации [*sender*]*_finished_key* вырабатывается согласно следующей формуле:

$$[sender]_finished_key = HKDF\text{-}Expand\text{-}Label(Finished\ Secret, "finished", "", HLen), \quad (7)$$

где *Finished Secret* — секретное значение, задаваемое в соответствии с 8.9, *HKDF-Expand-Label* — функция, определенная в 8.1.3.

Сторона, получившая сообщение Finished, должна проверить корректность содержащихся в сообщении данных и в случае их некорректности должна завершить работу протокола Handshake с оповещением decrypt_error (см. 7.2).

После отправки сообщений Finished и проверки корректности данных, содержащихся в полученном сообщении Finished, клиент и сервер могут начать обмен прикладными данными в рамках текущего соединения. При этом сервер может отправлять прикладные данные сразу после отправки сообщения Finished, не дожидаясь ответного сообщения клиента, в случае если текущее соединение соответствует соединению с односторонней аутентификацией.

Все записи, отправляемые после сообщения Finished, должны быть зашифрованы с помощью соответствующего ключевого материала трафика, выработанного на основе секретного значения [*sender*]*_application_traffic_secret_N*. В частности, это относится ко всем посылаемым оповещениям.

5.9 Post-handshake сообщения

Протокол TLS 1.3 позволяет отправлять сообщения после обмена main-handshake сообщениями в рамках протокола Handshake. Данные сообщения содержат набор данных, имеющих структуру Handshake, и зашифрованы с помощью соответствующего ключевого материала трафика, выработанного на основе секретного значения [*sender*]*_application_traffic_secret_N*.

5.9.1 Сообщение NewSessionTicket

В любое время после получения первого сообщения Finished от клиента сервер может послать клиенту сообщение NewSessionTicket, соответствующее некоторому значению *iPSK* и содержащее данные, необходимые для выработки значения *iPSK* из значения *RMS* (*resumption_master_secret*) на стороне клиента. Для краткости информацию, передаваемую в данном сообщении в поле ticket, будем называть тикетом.

Примечание — Несмотря на то, что секретное значение *RMS* зависит от первого сообщения Finished со стороны клиента, сервер, не запрашивающий аутентификацию клиента, может вычислить секретное значение *RMS*, самостоятельно сгенерировав строку *Finished*, соответствующую байтовому представлению первого сообщения Finished со стороны клиента, недостающую для хэш-функции *Transcript-Hash*. В этом случае сервер может отправить сообщение NewSessionTicket сразу после отправленного им сообщения Finished, не дожидаясь первого сообщения Finished со стороны клиента. Данные действия со стороны сервера могут быть уместны в случае, когда ожидается, что клиент откроет несколько параллельных соединений протокола TLS и выиграет от сокращения затрат вычислительных ресурсов на возобновление протокола Handshake.

Клиент может использовать значение *iPSK* для установления последующих соединений в рамках iPSK-only и iPSK-ECDHE режимов работы протокола Handshake, указав в поле identity расширения pre_shared_key значение поля ticket сообщения NewSessionTicket, соответствующего данному значению *iPSK* (см. подробнее 5.6.5).

Сервер может посылать не более 1024 сообщений NewSessionTicket в рамках одного соединения.

При этом тип аутентификации (двусторонняя или односторонняя), предоставляемый в рамках обладания значением *iPSK*, определяется типом аутентификации сторон в соединении на момент пересылки сообщения NewSessionTicket, ассоциированного с данным значением. Таким образом, значения *iPSK*, соответствующие сообщениям NewSessionTicket, пересылаемым до и после сообщений post-handshake аутентификации, будут обеспечивать разный тип аутентификации сторон в последующих соединениях.

Каждый тикет должен быть использован в рамках возобновления соединения только с криптонабором, поддерживающим тот алгоритм хэширования, который использовался в инициализирующем соединении.

Структура NewSessionTicket сообщения NewSessionTicket задается следующим образом:

```
struct {
    uint32 ticket_lifetime;
    uint32 ticket_age_add;
    opaque ticket_nonce<0..255>;
    opaque ticket<1..2^16-1>;
    Extension extensions<0..2^16-2>;
} NewSessionTicket;
```

где ticket_lifetime — срок жизни тикета в секундах, являющийся 32-битным значением, представленным в формате big-endian, и задающийся в соответствии с 5.9.1.2;

ticket_age_add — случайное 32-битное значение, которое используется для маскирования времени жизни тикета на стороне клиента.

Время жизни тикета на стороне клиента складывается с этим значением по модулю 2^{32}, и полученное значение передается в расширении pre_shared_key сообщения ClientHello (см. подробнее 5.6.5). Сервер должен генерировать новое значение для каждого отправляемого им тикета;

ticket_nonce — значение, которое является уникальным среди всех тикетов, выпущенных в рамках текущего соединения;

ticket — идентификатор (тикет) значения *iPSK*, формирующийся в соответствии с 5.9.1.1;

extensions — набор опциональных расширений. В случае если расширения не указываются, значение данного поля содержит пустую строку. Клиенты должны игнорировать нераспознанные расширения. В рамках настоящих рекомендаций не описываются расширения, которые могут использоваться в сообщении NewSessionTicket, и в

случае необходимости их использования их описание, исследование функционала, предоставляемого данными расширениями, а также анализ стойкости протокола должны проводиться отдельно.

Значение *iPSK*, ассоциированное с тикетом, вычисляется в соответствии с 8.6.

5.9.1.1 Формирование поля ticket сообщения NewSessionTicket

Поле ticket сообщения NewSessionTicket может содержать следующие данные:

- уникальный ключ поиска в базе данных тикетов сервера (аналог значения идентификатора сессии session_id, используемого в протоколе TLS 1.2, описанном в Р 1323565.1.020);

- данные, передаваемые клиенту в защищенном с помощью долговременного ключа сервера виде для того, чтобы не хранить на своей стороне параметры, необходимые для проверки данного тикета. Настоящие рекомендации не фиксируют данный способ формирования тикета. При необходимости использования данного способа его описание, исследование предоставляемого функционала, а также анализ стойкости протокола должны проводиться отдельно.

При формировании тикета в соответствии с первым способом, описанным выше, сервер должен хранить у себя базу данных тикетов, где каждому тикету соответствует следующий минимальный набор необходимых параметров:

а) время формирования тикета creation_time;

Примечание — Формат представления времени не фиксируется и выбирается на усмотрение сервером. Одним из вариантов представления может быть значение текущего времени в 32-битном формате UNIX (количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года).

б) значение ticket_lifetime, равное значению, указываемому в соответствующем сообщении NewSessionTicket;

в) значение ticket_age_add, равное значению, указываемому в соответствующем сообщении NewSessionTicket;

г) используемый алгоритм хэширования;

д) значение *iPSK*, соответствующее данному тикету;

е) допустимый режим использования данного значения *iPSK* (см. подробнее 5.6.6);

ж) тип аутентификации, соответствующий данному значению *iPSK* (двусторонняя или односторонняя);

и) значение global_expiration_time, равное одному из следующих значений:

1) наиболее ранней из дат окончания сроков действия сертификатов, ассоциированных с данным значением *iPSK* (см. 5.2);

2) дате окончания срока действия значения *ePSK*, ассоциированного с данным значением *iPSK* (см. 5.2);

Примечание — Формат представления времени должен соответствовать формату представления значения creation_time.

к) оставшееся количество попыток использования данного тикета (в случае, если максимально допустимое количество попыток, соответствующее политике безопасности сервера, превышает значение 1). В случае, когда в соединении, устанавливаемом в рамках режима работы протокола Handshake, использующего значение *iPSK*, сторонами было послано хотя бы одно сообщение, следующее за сообщением ServerHello, сервер должен уменьшить допустимое количество попыток использования данного значения *iPSK* на 1 вне зависимости от успешности установления данного соединения. В случае если оставшееся количество попыток использования достигло значения 0, сервер должен удалить из своей базы данных тикетов всю информацию, соответствующую данному тикету, и считать данный тикет недействительным.

В случае если используемое значение *iPSK* было выработано в результате цепочки соединений, где в рамках первоначального соединения клиент аутентифицировался за счет использования своего сертификата (в рамках ecdhe_ke схемы аутентифицированной выработки общего ключевого материала), серверу также рекомендуется проверять сертификаты в цепочке сертификатов на отозванность. Для этого серверу необходимо хранить в базе данных тикетов сертификат клиента в случае, если на момент формирования данного тикета клиент аутентифицировался с помощью использования данного сертификата. При этом в случае если в базе данных тикетов с тикетом, на основе которого устанавливалось текущее соединение, ассоциируется некоторый сертификат, то любой тикет, выработанный в ходе текущего соединения, также должен ассоциироваться с данным сертификатом.

Примечание — Тикет должен быть уникальным в рамках всех тикетов, хранимых в базе данных тикетов на текущий момент.

5.9.1.2 Формирование поля ticket_lifetime сообщения NewSessionTicket

Значение поля ticket_lifetime сообщения NewSessionTicket не должно превышать 604800 секунд (7 дней). Нулевое значение данного поля означает, что тикет должен быть сразу использован и удален. Клиенты не должны хранить тикеты дольше 7 дней независимо от значения поля ticket_lifetime и могут удалять тикеты раньше окончания установленного сервером срока жизни в соответствии с локальной политикой. Сервер может считать максимальным допустимым сроком жизни тикета значение, меньшее, чем указано в поле ticket_lifetime.

В соответствии с [1] и настоящими рекомендациями допускается создание новых тикетов в рамках соединения, установленного с помощью iPSK-ECDHE и iPSK-only режимов работы (режимов восстановления соединения). Такая возможность позволяет неограниченно продлевать срок жизни общего ключевого материала, выработанного в рамках изначального Full Handshake соединения. В связи с этим при работе протокола в соответствии с настоящими рекомендациями значение поля NewSessionTicket. ticket_lifetime не должно превышать минимума из следующих значений:

- 604800 секунд (7 дней);
- разницы между значениями переменных global_expiration_time и creation_time (см. 5.9.1.1);
- количества секунд, продиктованное другими локальными политиками сервера. Например, если к моменту создания тикета сертификат клиента, с помощью которого устанавливалось исходное соединение, отозван, то сервер может принять решение о запрете создания новых тикетов.

5.9.2 Post-handshake аутентификация

Если клиент готов аутентифицироваться после обмена main-handshake сообщениями (осуществить post-handshake аутентификацию), то он должен указать расширение post_handshake_auth в сообщении ClientHello. В случае получения данного расширения от клиента сервер может отправить запрос на аутентификацию клиента путем пересылки сообщения CertificateRequest (см. 5.7.2) в любой момент времени после получения main-handshake сообщения Finished со стороны клиента.

Если расширение post_handshake_auth не было указано в сообщении ClientHello, то сервер не должен запрашивать аутентификацию клиента после завершения протокола Handshake, в противном случае клиент должен ответить оповещением об ошибке unexpected_message (см. 7.2).

При получении CertificateRequest в рамках post-handshake аутентификации клиент должен ответить сообщениями аутентификации Certificate, CertificateVerify, Finished (см. 5.8). Если клиент отклоняет запрос на аутентификацию, то он должен ответить сообщениями Certificate иFinished, причем сообщение Certificate не должно содержать сертификатов (размер поля certificate_list должен быть равен нулю).

При проведении post-handshake аутентификации и формировании сообщений CertificateVerify и Finished в функцию *Transcript-Hash* подаются все main-handshake сообщения, переданные в рамках данного соединения, и только те post-handshake сообщения, которые определяют текущую процедуру проведения аутентификации (т. е. CertificateRequest и Certificate при формировании сообщения CertificateVerify и сообщения CertificateRequest, Certificate и CertificateVerify (если оно было послано) при формировании сообщения Finished).

Поскольку при аутентификации клиента может потребоваться взаимодействие с пользователем, сервер должен быть готов к получению произвольного количества сообщений между моментом отправки сообщения CertificateRequest и ответом клиента на данное сообщение. Кроме того, клиент, получивший несколько сообщений CertificateRequest, посланных сервером в близкой последовательности, может ответить на них в порядке, отличном от порядка, в котором сообщения CertificateRequest были посланы сервером, поскольку уникальное значение поля CertificateRequest.certificate_request_context позволяет серверу однозначно определить ответ клиента на запрос на аутентификацию.

5.9.3 Сообщение KeyUpdate

Сообщение KeyUpdate протокола Handshake посылается стороной (клиентом или сервером) для указания того, что отправляющая сторона обновляет ключевой материал трафика для формирования записи (значения [*sender*]_*write_key*, [*sender*]_*write_iv*, см. 8.4 и 6.3).

Данное сообщение может быть отправлено соответствующей стороной в любой момент времени после отправки ею main-handshake сообщения Finished. Если сообщение KeyUpdate было отправлено до main-handshake сообщения Finished, получающая его сторона должна завершить соединение с оповещением unexpected_message (см. 7.2).

Примечание — Сообщение KeyUpdate может быть послано сервером до получения main-handshake сообщения Finished со стороны клиента только в случае односторонней аутентификации.

Отправитель обязан посылать все последующие сообщения после передачи сообщения KeyUpdate, используя обновленный ключевой материал, вычисленный способом, описанным в 8.3. Получатель сообщения KeyUpdate должен обновить ключевой материал трафика для чтения записи (значения [*receiver*]_*read_key*, [*receiver*]_*read_iv*, см. 6.3).

Структура KeyUpdate сообщения KeyUpdate задается следующим образом:

```
enum {
    update_not_requested(0x00),
    update_requested(0x01),
    (0xFF)
} KeyUpdateRequest;

struct {
    KeyUpdateRequest request_update;
} KeyUpdate;
```

Поле request_update структуры KeyUpdate может принимать следующие значения:

- update_not_requested(0x00), в случае если отправитель не ожидает ответного сообщения KeyUpdate от получателя;
- update_requested(0x01), в случае если отправитель ожидает ответного сообщения KeyUpdate для обновления своего ключевого материала трафика для чтения записей.

В случае если в поле request_update записано значение update_requested, получатель должен отправить сообщение KeyUpdate с установленным значением update_not_requested до передачи дальнейших прикладных данных. Такой механизм позволяет обеим сторонам обновить ключевой материал трафика сразу для чтения и для записи. При этом, если сторона получила сразу несколько сообщений KeyUpdate с установленным значением update_requested, например, в период, когда эта сторона не отправляла никаких сообщений, она может ответить только одним сообщением KeyUpdate.

После отправки отправителем (*S*) сообщения $KeyUpdate^S$ получателю (*R*), где поле $KeyUpdate^S$.request_update принимает значение update_requested, отправитель должен быть готов получить произвольное количество сообщений до получения ответного сообщения $KeyUpdate^R$ от стороны *R*, поскольку эти сообщения могут быть отправлены *R* до получения сообщения $KeyUpdate^S$.

В случае если стороны независимо друг от друга и одновременно посылают сообщения KeyUpdate со значением поля request_update, равным update_requested, каждая из сторон должна отправить ответное сообщение, и обновление ключевого материала трафика для чтения и для записи произойдет дважды.

Отправитель и получатель должны зашифровывать сообщения KeyUpdate на необновленном ключевом материале трафика.

6 Протокол Record

Получив данные от протоколов более высокого уровня, протокол Record формирует из них последовательность структур, которые называются записями. Затем сформированные записи передаются транспортному протоколу.

Записи состоят из заголовка, описывающего передаваемые данные, и самих данных, которые передаются в открытом или защищенном виде в зависимости от текущего состояния соединения (см. подробнее 4.2). Заголовок всегда передается в открытом виде.

Данные, полученные от протокола транспортного уровня, протокол Record интерпретирует как записи и формирует из них сообщения для протоколов верхнего уровня, опираясь на заголовки записей.

Выделяют следующие типы данных, пересылаемых в рамках протокола Record:

```
enum {
        invalid(0x00),
        change_cipher_spec(0x14),
        alert(0x15),
        handshake(0x16),
        application_data(0x17),
        (0xFF)
} ContentType;
```

где change_cipher_spec — тип данных, отмененный в рамках версии протокола TLS 1.3 и сохраненный в целях поддержки режима совместимости;

alert — тип, соответствующий сообщениям протокола Alert;

handshake — тип, соответствующий сообщениям протокола Handshake;

application_data — тип, соответствующий сообщениям, содержащим прикладные данные.

6.1 Фрагментация

Принятые с верхнего уровня данные разбиваются протоколом Record на фрагменты, помещаемые в поля TLSPlaintext.fragment (см. подробнее 6.2). Значение поля TLSPlaintext.length не должно превышать 2^{14} (то есть размер поля TLSPlaintext.fragment не должен превышать 16 Кбайт). Если состояние соединения не подразумевает защиты данных, каждая запись задается структурой TLSPlaintext (см. подробнее 6.2). Если состояние соединения подразумевает защиту данных, каждая запись задается структурой TLSCiphertext. При этом размер защищенных данных, являющихся результатом работы AEAD алгоритма, может оказаться больше размера исходного фрагмента данных (это приводит к тому, что значение поля TLSCiphertext.length становится больше значения поля TLSPlaintext.length).

Примечание — При работе протокола в соответствии с криптонаборами, описанными в настоящих рекомендациях, значение поля TLSCiphertext.length не должно превышать $2^{14} + 1 + S$, где S — размер имитовставки в режиме MGM, определенный в 10.1.2.

Сообщения протокола Handshake могут быть разбиты на несколько фрагментов или объединены в один, при этом:

- сообщения протокола Handshake не могут объединяться с данными других типов в рамках одной записи;
- если сообщения протокола Handshake объединены в один фрагмент, в рамках работы протокола TLS 1.3 между этими сообщениями не должны пересылаться данные других типов;
- все сообщения протокола Handshake, объединенные в один фрагмент, должны быть обработаны в рамках одного и того же состояния соединения;
- запрещается посылать фрагменты нулевой длины для сообщений протокола Handshake.

Сообщения, содержащие прикладные данные, могут быть разбиты на несколько фрагментов или объединены в один, при этом:

- прикладные данные всегда пересылаются в защищенном виде и передаются протоколу Record без дополнительного форматирования;
- разрешается посылать фрагменты нулевой длины для сообщений, содержащих прикладные данные.

Сообщения протокола Alert не должны фрагментироваться. Запрещено объединять сообщения протокола Alert в один фрагмент.

Способы фрагментации не влияют на корректность работы всего протокола TLS 1.3 и должны определяться на этапе его реализации.

6.2 Формирование записи

Выделяют защищенные и незащищенные записи. Каждая запись содержит заголовок длины 5 байт, передаваемый в открытом виде, и фрагмент данных, передаваемых в открытом или защищенном виде. Заголовок состоит из трех полей: *type*, *legacy_record_version, length* в случае незащищенной записи и *opaque_ type*, *legacy_record_version, length* в случае защищенной. Поля заголовка указывают на тип сообщения, версию протокола и длину передаваемых данных соответственно. Поле *fragment*

незащищенной записи содержит фрагмент данных, передаваемых в открытом виде. Поле *encrypted_record* защищенной записи содержит фрагмент данных, передаваемых в защищенном виде. При этом с каждой записью неявно ассоциируется ее порядковый номер *seqnum* (см. подробнее 6.4).

В случае формирования незащищенной записи часть данных, полученных от протоколов более высокого уровня и выделенных при фрагментации в соответствии с 6.1, переводится в структуру TLSPlaintext, которая задается следующим образом:

```
struct {
    ContentType type;
    ProtocolVersion legacy_record_version;
    uint16 length;
    opaque fragment[TLSPlaintext.length];
} TLSPlaintext;
```

где
- type — тип сообщения длиной в 1 байт, указывающий на протокол верхнего уровня, используемый для обработки фрагмента данных, содержащегося в текущей записи;
- legacy_record_version — версия протокола длиной в 2 байта. Данное поле должно принимать значение 0x0303 для всех сообщений протокола TLS 1.3, за исключением исходного сообщения приветствия ClientHello (см. подробнее 5.5.1), то есть сообщения, отправленного до получения сообщения HelloRetryRequest, которое может также принимать значение 0x0301 в целях поддержки режима совместимости. Данное поле должно игнорироваться сторонами взаимодействия;
- length — длина фрагмента данных в байтах (2 байта). Данный параметр определяет количество байтов, передаваемых в записи после ее заголовка. Значение поля TLSPlaintext.length должно удовлетворять ограничениям, определенным в 6.1;
- fragment — фрагмент данных, передаваемых в открытом виде.

При формировании защищенной записи в соответствии с текущим состоянием соединения протокол Record вначале формирует данные в виде незащищенной структуры TLSPlaintext, а затем формирует из них данные защищенной структуры TLSCiphertext, которая задается следующим образом:

```
struct {
    ContentType opaque_type = application_data; /* 0x17 */
    ProtocolVersion legacy_record_version = 0x0303;
    uint16 length;
    opaque encrypted_record[TLSCiphertext.length];
} TLSCiphertext;
```

где
- opaque_type — тип сообщения длиной в 1 байт. Данное поле должно принимать значение 0x17, соответствующее типу application_data. После расшифрования данных поля encrypted_record структуры TLSCiphertext тип данных незащищенной записи содержится в поле TLSInnerPlaintext.type;
- legacy_record_version — версия протокола длиной в 2 байта. Данный параметр всегда должен иметь значение 0x0303;
- length — длина фрагмента данных в байтах (2 байта). Данный параметр определяет количество байтов, передаваемых в записи после ее заголовка. Значение поля TLSCiphertext.length должно удовлетворять ограничениям, определенным в 6.1;
- encrypted_record — фрагмент данных, передаваемых в защищенном виде и формирующийся из структуры TLSPlaintext в соответствии с согласованным криптонабором. Процесс формирования этого поля описан в 6.3.

6.3 Защита данных

В настоящем разделе все действия будут описаны для одной фиксированной стороны взаимодействия (клиента/сервера), которая обладает:

- ключевым материалом ([*sender*]_*write_key*, [*sender*]_*write_IV*) для формирования защищенных записей, принимающим значение (*client_write_key, client_write_IV*) для клиента, и значение (*server_write_key, server _write_IV*) для сервера;
- ключевым материалом ([*receiver*]_*read_key*, [*receiver*]_*read _IV* для обработки защищенных записей, принимающим значение [*server_write_key*, *server_write_IV*) для клиента, и значение (*client_write_key, client_write_IV*) для сервера.

Примечание — Режимы работы протокола TLS 1.3, описанные в настоящих рекомендациях, соответствуют симметричной схеме защиты данных: ключ шифрования и уникальный вектор, которые используются для формирования защищенной записи одной стороной, используются для расшифрования сообщения другой стороной.

Формирование защищенной записи из фрагмента данных, выделенного при фрагментации в соответствии с 6.1, выполняется в соответствии со следующими этапами, схематично отраженными на рисунке 6:

- формирование незащищенной записи, имеющей структуру TLSPlaintext;
- формирование структуры TLSInnerPlaintext;
- формирование значения *nonce* и *additional_data*;
- зашифрование данных структуры TLSInnerPlaintext с использованием ассоциированных данных *additional_data* и значения *nonce*;
- формирование защищенной записи, имеющей структуру TLSCiphertext.

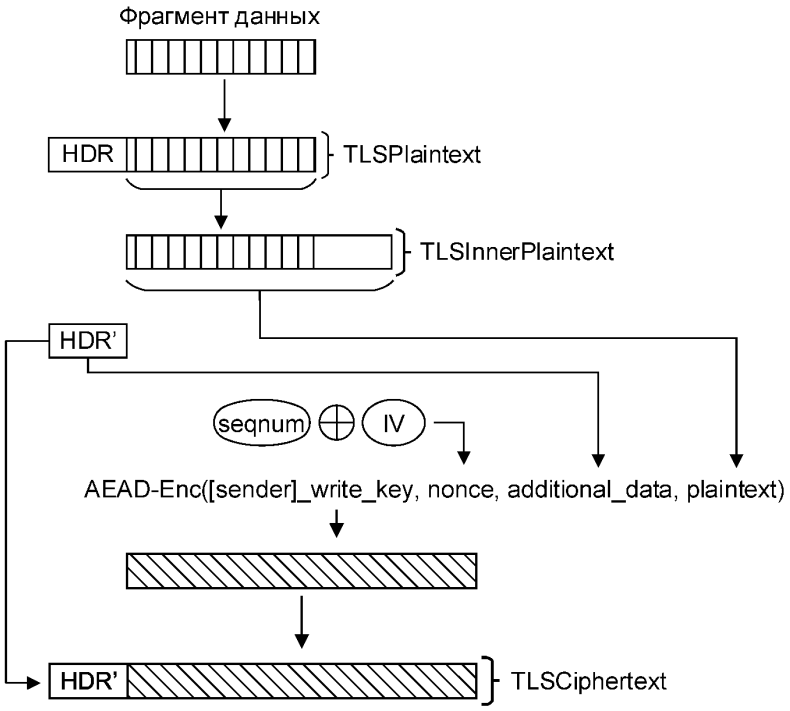


Рисунок 6 — Формирование защищенной записи

Примечание — На рисунке 6 под HDR и HDR' подразумеваются заголовки незащищенной и защищенной записей соответственно.

Для каждой формируемой записи с номером *seqnum* поле encrypted_record структуры TLSCiphertext содержит значение *ENCrecord*, которое формируется в соответствии с формулой:

$$ENCrecord = AEAD\text{-}Encrypt([sender]_write_key, nonce, additional_data, plaintext), \quad (8)$$

где:

- функция аутентифицированного шифрования *AEAD-Encrypt* задается согласованным криптонабором (см. 10.1, 10.1.2);
- значение *nonce* формируется из номера формируемой записи *seqnum* и уникального вектора [*sender*]_*write_IV* в соответствии с формулой:

$$nonce = STR_{IVLen}(seqnum) \oplus [sender]_write_IV; \quad (9)$$

- значение *additional_data* формируется в соответствии с формулой

$$additional_data = TLSCiphertext.opaque_type | TLSCiphertext.legacy_record_version | TLSCiphertext.length; \quad (10)$$

- значение *plaintext* содержит в себе данные структуры TLSInnerPlaintext, которая задается следующим образом:

```
struct {
      opaque content[TLSPlaintext.length];
      ContentType type;
      uint8 zeros[length_of_padding];
} TLSInnerPlaintext;
```

где content — значение поля TLSPLaintext.fragment;

type — значение поля TLSPLaintext.type;

zeros — поле, содержащее строку нулевых значений произвольной длины length_of_padding (см. 6.5).

Общий размер структуры TLSInnerPlaintext не должен превышать $2^{14} + 1$ байт.

Для каждой полученной записи с номером *seqnum* значение *plaintext* вычисляется из значения *ENCrecord* поля TLSCiphertext.encrypted_record в соответствии с формулой

$$plaintext = AEAD\text{-}Decrypt([receiver]_read_key, nonce, additional_data, ENCrecord); \quad (11)$$

где:

- значение *plaintext* содержит в себе данные структуры TLSInnerPlaintext, задающейся выше;
- функция расшифрования *AEAD-Decrypt* задается согласованным криптонабором (см. 10.1, 10.1.2);
- значение *nonce* формируется из номера получаемой записи *seqnum* и уникального вектора [*receiver*]_*write_IV* в соответствии с формулой

$$nonce = STR_{IVLen}(seqnum) \oplus [receiver]_write_IV; \quad (12)$$

- значение *additional_data* формируется в соответствии с формулой (10).

В случае если в рамках работы функции *AEAD-Decrypt* произошла ошибка, получающая сторона должна завершить соединение с оповещением bad_record_mac (см. 7.2).

6.4 Счетчик полученных/отправленных записей

Каждая из сторон взаимодействия ведет счетчик полученных и отправленных записей $seqnum^{read}$ и $seqnum^{write}$, каждый из которых может принимать значения от 0 до *SNMAX*-1 включительно. Таким образом, с каждой полученной/отправленной записью неявно ассоциируется 64-битный уникальный порядковый номер, соответствующий текущему значению счетчика $seqnum^{read}/seqnum^{write}$. Максимальное количество записей, которые могут передаваться в рамках одного значения [*sender*]_*write_key*, может быть меньше 2^{64} и может зависеть от выбранного криптонабора (см. подробнее 10.1.3).

В начале соединения, а также при смене ключевого материала трафика (см. 8.3) (при смене состояния соединения, см. подробнее 4.2) всем счетчикам присваивают нулевые значения. После получения/отправки очередной записи значение соответствующего счетчика увеличивается на 1.

6.5 Дополнение данных

Для того, чтобы скрыть размер передаваемой записи от стороннего наблюдателя, отправитель может использовать поле TLSInnerPlaintext.zeros, которое позволяет увеличить размер зашифровываемых данных. При этом отправитель сам решает, будет ли он дополнять данные или нет. В случае если сторона решает не дополнять данные, поле TLSInnerPlaintext.zeros должно содержать вектор нулевой длины. В случае дополнения поле TLSInnerPlaintext.zeros должно содержать только нулевые байты.

Для прикладных данных поле TLSInnerPlaintext.content может быть пустым (его размер может быть равен нулю). Стороны не должны отправлять записи типа handshake и alert, для которых поле TLSInnerPlaintext.content является пустым (содержит пустой вектор).

Дополнение данных автоматически проверяется механизмом защиты записи. После того, как получатель успешно расшифровал данные в поле TLSCiphertext.encrypted_record, осуществляется поиск первого ненулевого байта, начиная с конца полученных данных, который является типом записи. Указанный механизм также обеспечивает проверку того, что дополнение содержит только нулевые байты, что позволяет отследить ошибки при некорректном дополнении.

При реализации необходимо четко следить за тем, что поиск ненулевого байта должен ограничиваться тем массивом данных, который был возвращен в качестве результата расшифрования. В случае если получатель не находит ненулевой байт в расшифрованных данных, он должен завершить соединение протокола TLS с оповещением unexpected_message (см. 7.2).

Наличие дополнения не меняет ограничения на размер структуры TLSInnerPlaintext: общий размер указанной структуры не должен превышать 2^{14} + 1 байт.

7 Протокол Alert

Сообщение протокола Alert содержит информацию о пересылаемом оповещении и посылается в одном из следующих случаев:

- стороны хотят корректно завершить соединение (см. 7.1);
- при работе протокола TLS произошла ошибка (см. 7.2).

Информация об оповещении в сообщении протокола Alert определяется структурой Alert, задающейся следующим образом:

```
struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

Поле level указывает на уровень оповещения и задается следующим образом:

```
enum {
    warning(0x01),
    fatal(0x02),
    (0xFF)
} AlertLevel;
```

Поле description содержит тип оповещения, который задается следующим образом:

```
enum {
      close_notify(0x00),
      unexpected_message(0x0A),
      bad_record_mac(0x14),
```

record_overflow(0x16),
handshake_failure(0x28),
bad_certificate(0x2A),
unsupported_certificate(0x2B),
certificate_revoked(0x2C),
certificate_expired(0x2D),
certificate_unknown(0x2E),
illegal_parameter(0x2F),
unknown_ca(0x30),
access_denied(0x31),
decode_error(0x32),
decrypt_error(0x33),
protocol_version(0x46),
insufficient_security(0x47),
internal_error(0x50),
inappropriate_fallback(0x56),
user_canceled(0x5A),
missing_extension(0x6D),
unsupported_extension(0x6E),
unrecognized_name(0x70),
bad_certificate_status_response(0x71),
unknown_psk_identity(0x73),
certificate_required(0x74),
no_application_protocol(0x78),
(0xFF)
} AlertDescription;

Примечания

1 В соответствии с [1] в рамках работы протокола TLS 1.3 оповещения decryption_failed_RESERVED(0x15), decompression_failure_RESERVED(0x1E), no_certificate_RESERVED(0x29), export_restriction_RESERVED(0x3C) и no_renegotiation_RESERVED(0x64) пересылаться не должны. Клиенту и серверу рекомендуется поддерживать обработку данных оповещений с целью поддержки совместимости с более ранними версиями протокола TLS.

2 В рамках протокола TLS 1.3 уровень оповещения задается неявным образом с помощью типа пересылаемого оповещения, поэтому данное поле может быть проигнорировано. Все оповещения об ошибке должны отправляться с уровнем fatal и быть распознанными таковыми вне зависимости от значения поля level. Оповещения, не встречающиеся среди указанных в перечислении AlertDescription, должны быть распознаны как оповещения об ошибке.

7.1 Оповещения закрытия соединения

Для того, чтобы клиент и сервер могли корректно завершить соединение, отправляются следующие оповещения:

close_notify — оповещение, сигнализирующее о том, что его отправитель закрыл соединение для отправки данных и больше не отправит ни одного сообщения в рамках данного соединения. При этом указанные действия не влияют на чтение данных на стороне отправителя[1)].

[1)] Данное свойство является особенностью протокола TLS 1.3. В рамках работы протокола TLS версии ниже 1.3 при получении оповещения close_notify сторона взаимодействия должна была закрывать соединение для отправки данных, посылая ответное оповещение close_notify и удаляя все записи, ожидаемые к отправке, что вело к усечению при чтении данных на противоположной стороне.

Данное оповещение должно быть отправлено каждой стороной в случае, если ранее в процессе работы протокола TLS 1.3 не было послано ни одного оповещения об ошибке.

Любые данные, полученные после оповещения с типом close_notify, должны игнорироваться. В случае если закрытие соединения на транспортном уровне происходит до получения оповещения close_notify, получающая сторона не может быть уверена в получении всех данных, отправленных противоположной стороной взаимодействия.

Рекомендуется присваивать оповещению закрытия соединения close_notify уровень warning.

user_canceled — оповещение, сигнализирующее о том, что его отправитель завершает работу протокола Handshake по причине, не связанной с возникновением ошибок в рамках работы данного протокола. В указанном случае рекомендуется отправлять оповещение close_notify сразу после оповещения user_canceled.

В случае возникновения ошибки после завершения работы протокола Handshake по причине, не связанной с возникновением ошибок в рамках работы протокола TLS, стороне рекомендуется закрыть соединение только с отправкой оповещения close_notify.

Рекомендуется присваивать оповещению user_canceled уровень warning.

7.2 Оповещения об ошибках

В случае возникновения ошибки в процессе работы протокола TLS сторона взаимодействия посылает оповещение об ошибке. Все оповещения об ошибке, указанные в настоящем разделе, имеют уровень fatal. При получении оповещения об ошибке сторона должна отправить в ответ соответствующее оповещение и закрыть соединение для чтения и записи данных.

Настоящие рекомендации определяют оповещения об ошибке в соответствии с таблицами 3—5.

Таблица 3 — Оповещения об ошибке, которые могут возникнуть в процессе работы протокола Handshake

Название оповещения	Описание оповещения
handshake_failure	Отправитель не смог согласовать необходимые параметры безопасности
illegal_parameter	Поле сообщения протокола Handshake некорректно, либо не согласуется с другими полями. Данное оповещение используется для указания ошибок, связанных с некорректным значением
unknown_ca	Сертификат удостоверяющего центра (УЦ) не был найден или не совпал ни с одним из известных сертификатов доверенных УЦ
access_denied	Был получен корректный сертификат, однако при применении политики контроля доступа отправитель решил не продолжать согласование соединения
decrypt_error	При выполнении криптографической операции (например, при проверке подписи, проверке содержимого сообщения Finished или проверке binder-значения) во время работы протокола Handshake возникла ошибка
protocol_version	Работа по указанной версии протокола не поддерживается
insufficient_security	Посылается вместо оповещения handshake_failure в том случае, если сервер требует криптонаборы, обеспечивающие более высокий уровень стойкости, чем те, что поддерживаются клиентом
unsupported_ extension	Посылается сторонами в случае, если было получено сообщение протокола Handshake, содержащее расширение, использование которого запрещено. Кроме того, данное оповещение посылается в случае, если в сообщении ServerHello или Certificate содержится расширение, не указанное ранее в сообщениях ClientHello или CertificateRequest соответственно

Окончание таблицы 3

Название оповещения	Описание оповещения
bad_certificate	Целостность сертификата нарушена (например, сертификат поврежден или изменен), сертификат содержит некорректную подпись и т. п.
unsupported_ certificate	Был получен сертификат неподдерживаемого типа
certificate_revoked	Сертификат был отозван подписывающей стороной
certificate_expired	Срок действия сертификата истек или сертификат не действует в настоящее время
certificate_unknown	При обработке сертификата возникла ошибка, не соответствующая ни одному из вышеперечисленных случаев
inappropriate_fallback	Отправляется сервером в случае обнаружения несанкционированного понижения используемой версии протокола TLS (см. подробнее 11.2.2)
missing_extension	Отправляется в случае, если полученное сообщение протокола Handshake не содержит расширение, являющееся обязательным в рамках текущего режима работы протокола
unrecognized_name	Отправляется сервером в случае, если не существует виртуального сервера, соответствующего имени, указанному клиентом в расширении server_name
bad_certificate_ status_response	Отправляется клиентом в случае, если сервер указал некорректный OCSP ответ в расширении status_request
unknown_psk_identity	Отправляется сервером в случае, если он желает установить соединение в рамках psk_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала, однако клиент не предоставил ни одного корректного PSK-значения. Данное оповещение об ошибке является опциональным и вместо него сервер может отправить оповещение decrypt_error
certificate_required	Отправляется сервером в случае, если требуется аутентификация клиента, но клиент не прислал ни одного сертификата
no_application_ protocol	Отправляется сервером в случае, если клиент не указал в расширении application_layer_ protocol_negotiation ни одного протокола, поддерживаемого сервером

Таблица 4 — Оповещения об ошибке, которые могут возникнуть в процессе работы протокола Record

Название оповещения	Описание оповещения
bad_record_mac	Данное оповещение посылается в том случае, если при расшифровании полученной защищенной записи в рамках работы функции *AEAD-Decrypt* возникла ошибка
record_overflow	Данное оповещение посылается в том случае, если длина полученной или расшифрованной записи превышает максимально допустимое значение

Таблица 5 — Общие оповещения об ошибке, которые могут возникнуть в процессе работы протокола TLS 1.3

Название оповещения	Описание оповещения
unexpected_message	Было получено некорректное сообщение
decode_error	Сообщение не может быть обработано, так как содержит одно или несколько некорректных полей. Данное оповещение используется для указания ошибок, связанных с синтаксисом протокола TLS
internal_error	Произошла внутренняя ошибка, не связанная с работой протокола (например, ошибка выделения памяти), которая делает невозможным продолжение дальнейшей работы протокола

8 Криптографические вычисления

8.1 Функции, используемые при выработке ключей

8.1.1 Функция *HKDF-Extract*

Функция *HKDF-Extract* задается следующим образом.

Входные аргументы:

- $Salt \in B_s$, $s \geq 0$, опциональный аргумент; если данный аргумент не был подан на вход функции *HKDF-Extract*, то его значение устанавливается равным 0^{HLen};
- $IKM \in B_s$, $s \geq 0$, входной ключевой материал.

Результат работы:

- $PRK \in B_{HLen}$, псевдослучайный ключ.

Функция *HKDF-Extract* задается в соответствии со следующей формулой:

$$HKDF\text{-}Extract\ (Salt, IKM) = HMAC(Salt, IKM) = PRK. \tag{13}$$

8.1.2 Функция *HKDF-Expand*

Функция *HKDF-Expand* задается следующим образом.

Входные аргументы:

- $PRK \in B_s$, $s \geq HLen$, псевдослучайный ключ;
- $info \in B_s$, $s \geq 0$, строка с опциональными контекстными данными;
- $L \leq 255 \cdot HLen$, байтовая длина ключевого материала, являющегося результатом работы функции *HKDF-Expand*.

Результат работы:

- $OKM \in B_L$, ключевой материал длины L.

Функция *HKDF-Expand* задается в соответствии со следующей формулой:

$$\begin{aligned} I &= \lceil L/HLen \rceil, \\ T &= T_1 | T_2 | \ldots | T_I, \end{aligned} \tag{14}$$

где:

$$T_i = HMAC\ (PRK, T_{i-1} | info | i),\ i \in \{1, \ldots, I\},\ T_0 \in B_0,$$

$$HKDF\text{-}Expand\ (PRK, info, L) = LMB_L(T) = OKM.$$

Примечание — В формуле (14) при вычислении значения функции HMAC параметр i конкатенируется с остальными байтовыми строками без дополнительного форматирования, поскольку принимаемые им значения представимы в виде 1 байта.

8.1.3 Функция *HKDF-Expand-Label*

Функция *HKDF-Expand-Label* задается следующим образом.

Входные аргументы:

- $Secret \in B_s$, $s \geq HLen$, псевдослучайный ключ;
- $Label \in B_s$, $1 \leq s \leq 249$, строковая константа;
- $Context \in B_s$, $0 \leq s \leq 255$, строка с опциональными контекстными данными;
- $Length \leq 255 \cdot HLen$, длина ключевого материала в байтах, являющегося результатом работы функции *HKDF-Expand-Label*.

Результат работы:

- ключевой материал длины *Length*.

Функция *HKDF-Expand-Label* задается в соответствии со следующей формулой:

$$\begin{aligned} &HKDF\text{-}Expand\text{-}Label\ (Secret, Label\ Context, Length) = \\ &HKDF\text{-}Expand\ (Secret, HkdfLabel, Length), \end{aligned} \tag{15}$$

где значение *HkdfLabel* задается следующей структурой:

```
struct {
    uint16 length = Length;
    opaque label<7..255> = "tls13" + Label;
    opaque context<0..255> = Context;
} HkdfLabel;
```

8.1.4 Функция *Derive-Secret*

Функция *Derive-Secret* задается следующим образом.

Входные аргументы:

- $Secret \in B_s$, $s \geq HLen$, псевдослучайный ключ;
- $Label \in B_s$, $1 \leq s \leq 249$, строковая константа;
- $Messages = \{M_1, \ldots, M_n\}$, $M_i \in B_{s_i}$, где $i \in \{1, \ldots, n\}$, $s_1 \geq 0$ при $n = 1$, $s_i > 0$ при $n > 1$.

Результат работы:

- ключевой материал длины *HLen*.

Функция *Derive-Secret* задается в соответствии со следующей формулой:

$$Derive\text{-}Secret\ (Secret, Label, Messages) = HKDF\text{-}Expand\text{-}Label\ (Secret, Label, Transcript\text{-}Hash\ (Messages), HLen). \quad (16)$$

8.2 Иерархия ключей

В рамках работы протокола TLS 1.3 можно выделить следующую ключевую иерархию:

а) энтропийные данные *PSK* (значение *iPSK* или *ePSK*, вырабатываемое в соответствии с 8.6) и *ECDHE* (вырабатываемое в соответствии с 8.5), с помощью которых вырабатываются значения *EarlySecret, HandshakeSecret, MasterSecret*;

б) секретные значения (значения *Secret*), перечисленные в таблице 6:

Таблица 6 — Секретные значения *Secret*

Секретное значение *Secret*	Краткое обозначение
binder_secret	*BS*
client_handshake_traffic_secret	*CHTS*
server_handshake_traffic_secret	*SHTS*
client_application_traffic_secret_N, N ≥ 0	$CATS_N$
server_application_traffic_secret_N, N ≥ 0	$SATS_N$
exporter_master_secret	*EMS*
resumption_master_secret	*RMS*

Примечание — В таблице 6 отсутствуют секретные значения *client_early_traffic_secret* и *early_exporter_master_secret*, описанные в [1], так как в версии протокола TLS 1.3, соответствующей настоящим рекомендациям, пересылка 0-RTT данных запрещена.

в) ключевой материал, вырабатываемый из секретных значений *Secret*:

1) ключ *HMAC_binder_key*, используемый для формирования binder-значения (см. 5.6.5.3);
2) значение *exporter_value* (см. 8.7).

Примечание — Настоящие рекомендации не определяют функцию TLS-Exporter, используемую в [1] для формирования значения *exporter_value*, и значения ее аргументов *label* и *context_value* (см. рисунок 7);

3) ключевой материал трафика, состоящий из ключей [*sender*]*_write_key* и вектора инициализации [*sender*]*_write_iv*, используемый для защиты записей в протоколе Record (см. 8.4);
4) ключи [*sender*]*_finished_key*, используемые для формирования данных verify_data сообщения Finished (см. 5.8.3).

Описанная ключевая иерархия представлена в виде схемы на рисунке 7. Ее основной принцип заключается в следующем: энтропийные данные, обозначенные в левой части схемы, не зависят от сообщений протокола Handshake, пересылаемых в рамках установления текущего соединения, в то время как секретные значения множества *Secret*, находящиеся в правой части схемы, зависят от сообщений протокола Handshake, пересылаемых в рамках установления текущего соединения, поэтому могут использоваться для выработки ключевого материала трафика.

Секретные значения множества *Secret* формируются с помощью энтропийных данных (*ECDHE* и *PSK*), подающихся на вход функциям *HKDF-Extract* и *Derive-Secret* в соответствии со схемой, представленной на рисунке 7. Если одно из значений *PSK* или *ECDHE* не определено в момент выработки ключа обработки записей, то подразумевается, что вместо него на вход функции *HKDF-Extract* подается строка 0^{HLen}. При этом одновременное отсутствие энтропийных значений *PSK* и *ECDHE* не допускается в рамках работы протокола TLS 1.3.

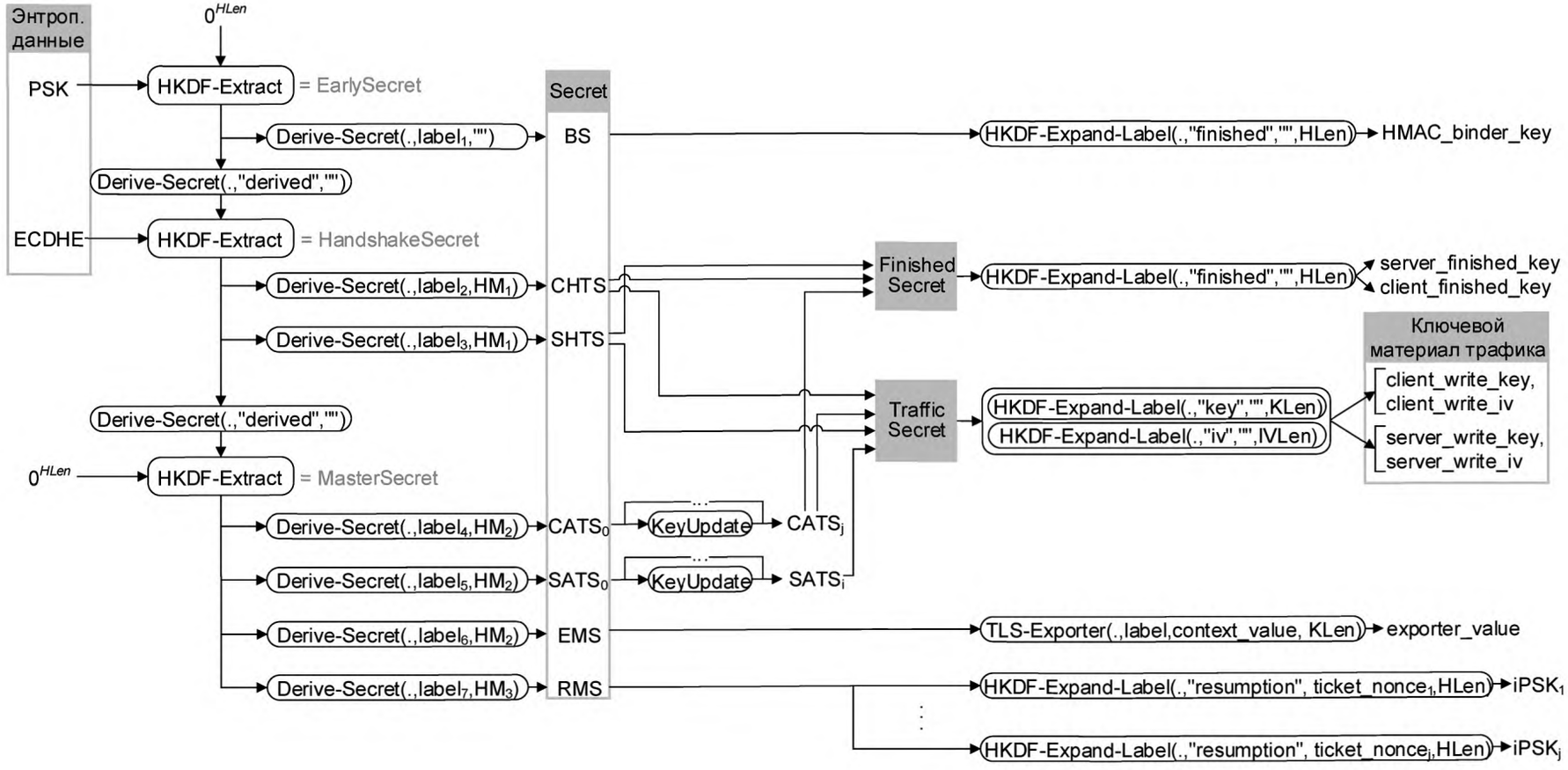


Рисунок 7 — Иерархия ключей

Примечания

1 На рисунке 7 подразумевается, что для функции *HKDF-Extract* (см. 8.1.1) аргумент *Salt* содержит данные, поступившие сверху, аргумент *IKM* содержит данные, поступившие слева, а результат работы данной функции направляется вниз, при этом обозначение данного результата приводится справа.

2 На рисунке 7 под обозначением *BS* (*binder_secret*) подразумевается обозначение *binder_key*, используемое в [1].

3 Значение *EarlySecret* определяется конкретным значением *PSK*. Таким образом, до момента выбора сервером конкретного тикета, соответствующего значению *PSK*, из списка, предложенного клиентом, клиент должен вычислять значение *EarlySecret* для каждого из значений *PSK*, соответствующих предложенным им значениям тикетов.

Аргументами функции *Derive-Secret*, помимо значений, являющихся результатом функции *HKDF-Extract*, являются метки $label_i$, $1 \le i \le 7$ (см. таблицу 7) и упорядоченные множества строк HM_j, $1 \le j \le 3$ (см. таблицу 8).

Таблица 7 — Задание меток $label_i$

Метка	Значение
$label_1$	*"res binder"/"ext binder"*
$label_2$	*"c hs traffic"*
$label_3$	*"s hs traffic"*
$label_4$	*"c ap traffic"*
$label_5$	*"s ap traffic"*
$label_6$	*"exp master"*
$label_7$	*"res master"*
Примечание — Под обозначением «*"res binder"/"ext binder"*» в данной таблице подразумевается одна из двух строковых констант: *"res binder"*, если секретное значение *BS* формируется с помощью значения *iPSK*, и *"ext binder"*, если секретное значение *BS* формируется с помощью значения *ePSK*.	

Таблица 8 — Задание множеств HM_j

Множество	Содержимое
HM_1	{*ClientHello*, ... , *ServerHello*}
HM_2	{*ClientHello*, ... , *Finished* со стороны сервера}
HM_3	{*ClientHello*, ... , *Finished* со стороны клиента}

8.3 Обновление секретных значений

После завершения работы протокола Handshake обе стороны могут инициировать обновление текущего секретного значения [*sender*]*_application_traffic_secret_N* из множества *Secret*, используя сообщение KeyUpdate, определенное в 5.9.3.

Новое значение [*sender*]*_application_traffic_secret_*(*N* + 1) вырабатывается из текущего значения [*sender*]*_application_traffic_secret_N* в соответствии со следующей формулой:

$$[sender]_application_traffic_secret_(N + 1) = HKDF\text{-}Expand\text{-}Label([sender]_application_traffic_secret_N, "traffic\ upd", "", HLen). \quad (17)$$

Сразу после выработки нового значения [*sender*]*_application_traffic_secret_*(*N* + 1) прежнее значение [*sender*]*_application_traffic_secret_*(*N*) и ассоциированный с ним ключевой материал трафика (см. 8.4) должны быть удалены.

8.4 Ключевой материал трафика

Ключевой материал трафика [*sender*]_*write_key*, [*sender*]_*write_iv* вычисляется следующим образом:

$$\begin{aligned} [sender]_write_key &= HKDF\text{-}Expand\text{-}Label\,(Traffic\ Secret, "key", "", KLen), \\ [sender]_write_iv &= HKDF\text{-}Expand\text{-}Label\,(Traffic\ Secret, "iv", "", IVLen), \end{aligned} \quad (18)$$

где секретное значение *Traffic Secret* зависит от этапа состояния соединения, на котором происходит обработка данных. Возможные значения *Traffic Secret* указаны в таблице 9.

Таблица 9 — Соответствие значения *Traffic Secret* этапу состояния соединения

Этап состояния соединения	Значение *Traffic Secret*
Этап выработки параметров соединения и аутентификации	[*sender*]_*handshake_traffic_secret*
Этап пересылки прикладных данных и post-handshake сообщений	[*sender*]_*application_traffic_secret_N*

При изменении значения *Traffic Secret* [при обновлении ключевого материала с помощью сообщения KeyUpdate (см. 8.3) или при переходе между этапами состояния соединения] ключевой материал трафика должен перевычисляться.

8.5 Выработка общего секретного значения *ECDHE*

Общее секретное значение *ECDHE* вырабатывается сторонами в случае установления соединения в рамках ecdhe_ke или psk_ecdhe_ke схемы аутентифицированной выработки общего ключевого материала в ходе обмена сообщениями ClientHello, ServerHello, HelloRetryRequest в соответствии с 8.5.1 и 8.5.2.

Примечания

1 Операции над точками эллиптических кривых задаются в соответствии с ГОСТ Р 34.10.

2 Описанный в разделах 8.5.1 и 8.5.2 порядок выработки секретного значения *ECDHE* на стороне клиента и на стороне сервера применим только для кривых, перечисленных в разделе 10.3. В случае появления в будущем новых стандартизованных кривых указанный порядок подлежит пересмотру и возможным изменениям.

8.5.1 Выработка значения *ECDHE* на стороне клиента

Клиент вырабатывает значение *ECDHE* следующим образом:

а) из списка поддерживаемых клиентом кривых $E_1, \ldots, E_R$ выбирает набор кривых $E_{i_1}, \ldots, E_{i_r}$, $1 \le i_j \le i_r \le R$, где:

1) $r \ge 1$ в случае первичной отправки сообщения;

2) $r = 1$ в случае ответа на сообщение HelloRetryRequest, при этом E_{i1} соответствует кривой, указанной в расширении key_share сообщения HelloRetryRequest (см. 5.6.4.2);

б) генерирует эфемерные ключевые пары $(d^{i_1}{}_c, Q^{i_r}{}_c), \ldots, (d^{i_1}{}_c, Q^{i_r}{}_c)$, соответствующие кривым $E_{i_1}, \ldots, E_{i_r}$, где для любого $i \in \{i_1, \ldots, i_r\}$:

1) $d^i{}_c$ выбирается случайно из множества $\{1, \ldots, q_i - 1\}$;

2) $Q^i{}_c = d^i{}_c \cdot P_i$;

в) отправляет серверу сообщение ClientHello, сформированное в соответствии с 5.5.1, содержащее:

1) расширение key_share со значениями открытых эфемерных ключей $Q^{i_1}{}_c, \ldots, Q^{i_r}{}_c$, сформированное в соответствии с 5.6.4.1;

2) расширение supported_groups со списком поддерживаемых кривых $E_1, \ldots, E_R$, сформированное в соответствии с 5.6.3

и переходит в состояние ожидания ответа от сервера;

г) в случае получения сообщения HelloRetryRequest переходит на первый шаг [перечисление а) 8.5.1], корректируя параметры в соответствии с 5.5.1; в случае получения сообщения ServerHello переходит к пятому шагу [перечисление д) 8.5.1]; в противном случае завершает соединение с оповещением об ошибке unexpected_message(см. 7.2);

д) извлекает из полученного сообщения ServerHello эллиптическую кривую E_{res} и открытый эфемерный ключ $Q^{res}{}_s$, $res \in \{1, \ldots, R\}$. Проверяет, что значение $Q^{res}{}_s$ принадлежит кривой E_{res}. В слу-

чае если это условие не выполнено, клиент должен завершить работу протокола TLS с оповещением handshake_failure;

е) формирует значение Q^{ECDHE} в соответствии со следующей формулой:

$$Q^{ECDHE} = (X^{ECDHE}, Y^{ECDHE}) = (h_{res} \cdot d^{res}_{c}) \cdot Q^{res}_{s}; \quad (19)$$

ж) проверяет, что $Q^{ECDHE} \neq O_{res}$. В случае если это условие не выполнено, клиент должен завершить работу протокола TLS с оповещением уровня handshake_failure;

и) значение *ECDHE* является байтовым представлением координаты X^{ECDHE} точки Q^{ECDHE} в формате little-endian, т. е. формируется в соответствии со следующей формулой:

$$ECDHE = str_{coordinate_length}(X^{ECDHE}), \quad (20)$$

где значение параметра *coordinate_length* определяется в соответствии с таблицей 17.

8.5.2 Выработка значения *ECDHE* на стороне сервера

Сервер, получив сообщение ClientHello, вырабатывает значение *ECDHE* следующим образом:

а) из списка $E_1, \ldots, E_R$, указанного клиентом в расширении supported_groups, выбирает кривую E_{res}, $res \in \{1, \ldots, R\}$, и соответствующее ей значение открытого эфемерного ключа Q^{res}_{c} из списка значений $Q^{i_1}{}_{c}, \ldots Q^{i_r}{}_{c}$, $1 \leq i_1 \leq i_r \leq R$, указанных в расширении key_share. В случае если такой эфемерный ключ не найден (т. е. $res \in \{1, \ldots, R\}/\{i_1, \ldots, i_r\}$), сервер может инициировать процедуру пересогласования открытых эфемерных ключей (см. 5.4), в рамках которой он отправит сообщение HelloRetryRequest, сформированное в соответствии с 5.5.3, содержащее расширение key_share, сформированное в соответствии с 5.6.4.2, содержащее информацию о кривой E_{res}, и перейдет в состояние ожидания сообщения ClientHello от клиента;

б) проверяет, что значение Q^{res}_{c} принадлежит кривой E_{res}. В случае если это условие не выполнено, сервер должен завершить работу протокола TLS с оповещением уровня handshake_failure;

в) генерирует эфемерную ключевую пару (d^{res}_{s}, Q^{res}_{s}) соответствующую кривой E_{res}:

1) d^{res}_{s} выбирается случайно из множества $\{1, \ldots, q_{res} - 1\}$;

2) $Q^{res}_{s} = d^{res}_{s} \cdot P_{res}$;

г) отправляет клиенту сообщение ServerHello, сформированное в соответствии с 5.5.2, содержащее расширение key_share со значением открытого эфемерного ключа Q^{res}_{s}, соответствующего кривой E_{res}, сформированное в соответствии с 5.6.4.3;

д) формирует значение Q^{ECDHE} в соответствии со следующей формулой:

$$Q^{ECDHE} = (X^{ECDHE}, Y^{ECDHE}) = (h_{res} \cdot d^{res}_{s}) \cdot Q^{res}_{c}; \quad (21)$$

е) проверяет, что $Q^{ECDHE} \neq O_{res}$. В случае если это условие не выполнено, сервер должен завершить работу протокола TLS с оповещением handshake_failure;

ж) значение *ECDHE* является байтовым представлением координаты X^{ECDHE} точки Q^{ECDHE} в формате little-endian, т. е. формируется в соответствии со следующей формулой:

$$ECDHE = str_{coordinate_length}(X^{ECDHE}), \quad (22)$$

где значение параметра *coordinate_length* определяется в соответствии с таблицей 17.

8.6 Выработка предварительно распределенного секрета *PSK*

Значение *PSK*, соответствующее тикету, указываемому в поле PskIdentity.identity расширения pre_shared_key сообщения ClientHello, или значению поля selected_identity расширения pre_shared_key сообщения ServerHello, формируется в соответствии с одним из следующих способов в зависимости от типа предварительно распределенного секрета:

а) значение внутреннего предварительно распределенного секрета *iPSK*, который ассоциируется с тикетом, пересылаемым в сообщении NewSessionTicket (см. 5.9.1), вычисляется следующим образом:

$$iPSK = HKDF - Expand - Label(RMS, «resumption», ticket_nonce, HLen), \quad (23)$$

где значение *RMS* задается в соответствии с 8.2;

значение *ticket_nonce* соответствует полю ticket_nonce соответствующего сообщения NewSessionTicket (см. 5.9.1).

Примечание — Поскольку значение поля ticket_nonce должно быть уникальным для каждого сообщения NewSessionTicket, отправленного в рамках одного соединения, для каждого тикета будет формироваться новое секретное значение *iPSK*;

б) значение внешнего предварительно распределенного секрета *ePSK* вычисляется в соответствии с механизмом, который не фиксируется в настоящем документе и который, в случае использования данного типа предварительно распределенного секрета, должен быть описан и исследован отдельно.

8.7 Экспорт ключевого материала

Значение *exporter_value* используется для экспорта ключевого материала из протокола TLS в протокол, находящийся на верхнем уровне. Формирование значения *exporter_value* описано в [1] в разделе 7.5; формат меток, используемых для формирования *exporter_value*, описан в [2] в разделе 4.

Примечание — Настоящие рекомендации не фиксируют механизм использования данного значения в рамках вышестоящих протоколов; описание данного механизма, исследование функционала, предоставляемого в результате использования данного значения, а также анализ стойкости протокола, использующего данное значение, должны проводиться отдельно.

8.8 Функция *Transcript-Hash*

Функция *Transcript-Hash* задается следующим образом:

$$Transcript\text{-}Hash\ (M_1, M_2, \dots, M_n) = HASH\ (M^*_1|M_2| \dots |M_n), \quad (24)$$

где:

а) $M_i \in B_{s_i}$, где:

1) $s_1 \geq 0$, если $n = 1$;

2) $s_i > 0$, $i \in \{1, \dots, n\}$, если $n > 1$.

Примечание — При использовании функции *Transcript-Hash* в протоколе TLS 1.3 множество строк M_1, M_2, ... , M_n либо содержит пустую строку (состоит из единственного элемента $M_1 \in B_0$), либо состоит из множества строк, соответствующих байтовым представлениям посланных или части формируемых (см. 5.6.5.3) сообщений протокола Handshake.

б) M^*_1 задается следующим образом:

1) если $n \geq 2$, $M_2 = HelloRetryRequest$ (т. е. если в рамках текущего соединения было послано сообщение HelloRetryRequest), то

$$M^*_1 = message_hash|0x00|0x00|HLen|HASH(M_1), \quad (25)$$

где *message_hash* — значение кода типа сообщения протокола Handshake, определенное в 5.1;

2) $M^*_1 = M_1$, в противном случае.

Примечание — В целях эффективности реализации значение функции *Transcript-Hash* может вычисляться постепенно по мере поступления сообщений в рамках работы протокола Handshake.

8.9 Значения *Handshake Context* и *Finished Secret*

Значение *Handshake Context* представляет собой упорядоченное множество строк, являющихся байтовым представлением сообщений протокола Handshake, которое зависит от текущего режима аутентификации и задается в соответствии с таблицей 10.

Значение *Finished Secret* задается в соответствии с режимами аутентификации (см. таблицу 10) и используется для формирования ключа вычисления кода аутентификации (см. 5.8.3).

Таблица 10 — Задание значений *Handshake Context* и *Finished Secret*

Режим аутентификации	Значение *Handshake Context*	Значение *Finished Secret*
аутентификация сервера	*{ClientHello, ..., EncryptedExtensions/CentificateRequest}*	*SHTS*
main-handshake аутентификация клиента	*{ClientHello, ..., Finished* со стороны сервера}	*CHTS*
post-handshake аутентификация клиента	*{ClientHello, ...,* первый *Finished* со стороны клиента, *CertificateRequest}*	$CATS_N$

Примечания

1 Под обозначением «*EncryptedExtensions/CertificateRequest*» в данной таблице подразумевается строка, соответствующая байтовому представлению последнего из полученных сообщений EncryptedExtensions или CertificateRequest соответственно.

2 Значения *Finished Secret* приводятся в кратком обозначении (см. подробнее 8.2).

9 Прикладные данные

Прикладные данные всегда пересылаются в защищенном виде, передаются протоколу Record без дополнительного форматирования и имеют тип application_data(0x17)(см. 5.1).

Не рекомендуется передавать прикладные данные сразу после передачи сообщения Finished со стороны сервера в случае двусторонней аутентификации клиента и сервера. На момент передачи прикладных данных непосредственно после сообщения Finished со стороны сервера клиент не является аутентифицированным, поэтому при неправильной работе приложения прикладного уровня могут быть скомпрометированы конфиденциальные данные.

10 Использование российских криптографических алгоритмов

10.1 Идентификаторы криптонаборов из реестра «TLSCipherSuites»

Настоящие рекомендации вводят следующие идентификаторы IANA из приватной области реестра «TLSCipherSuites», указываемые в сообщениях ClientHello и ServerHello:

Таблица 11 — Идентификаторы криптонаборов реестра «TLSCipherSuites»

Наименование криптонабора	Номер криптонабора
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_L	{0xC1, 0x03}
TLS_GOSTR341112_256_WITH_MAGMA_MGM_L	{0xC1, 0x04}
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_S	{0xC1, 0x05}
TLS_GOSTR341112_256_WITH_MAGMA_MGM_S	{0xC1, 0x06}

Каждый из вышеперечисленных идентификаторов определяет криптонабор, предназначенный для использования в протоколе TLS 1.3 и задающий криптографические параметры (блочный шифр, параметры ключевого дерева и ограничения числа передаваемых записей) в соответствии с 10.1.1—10.1.4.

Указанный порядок следования криптонаборов является рекомендуемым порядком предпочтения для клиентов, поддерживающих работу со всеми криптонаборами.

Примечание — Рекомендации по использованию описанных криптонаборов в СКЗИ в зависимости от области их применения приводятся в приложении А.

10.1.1 Блочный шифр

Определенные в настоящих рекомендациях криптонаборы в качестве блочного шифра используют шифр «Магма» или «Кузнечик», определенный в ГОСТ Р 34.12. Длина блока составляет 16 байт (n = 16) для шифра «Кузнечик» и 8 байт (n = 8) для шифра «Магма», длина ключей в обоих случаях составляет 32 байта (*KLen* = 32).

Таблица 12 — Используемые блочные шифры

Криптонабор	Блочный шифр
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_L	«Кузнечик» (ГОСТ Р 34.12)
TLS_GOSTR341112_256_WITH_MAGMA_MGM_L	«Магма» (ГОСТ Р 34.12)
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_S	«Кузнечик» (ГОСТ Р 34.12)
TLS_GOSTR341112_256_WITH_MAGMA_MGM_S	«Магма» (ГОСТ Р 34.12)

10.1.2 AEAD алгоритм

Определенные в настоящих рекомендациях криптонаборы в качестве AEAD алгоритма используют режим MGM работы блочного шифра, описанный в Р 1323565.1.026, с длиной имитовставки *S*, введенной в Р 1323565.1.026, равной *n*. Параметр *IVLen* принимает значение *n*.

При этом для каждой формируемой записи с номером *seqnum* функция зашифрования *AEAD-Encrypt* задается следующим образом.

Входные аргументы:

- $K \in B_{KLen}$, ключ шифрования;
- $nonce \in B_{IVLen}$, уникальный вектор;
- $A \in B_s$, $s \geq 0$, дополнительные имитозащищаемые данные;
- $P \in B_s$, $s \geq 0$, открытый текст.

Результат работы:

- $C|T$, где $C \in B_{|P|}$ — шифртекст, $T \in B_s$ — имитовставка.

Функция зашифрования *AEAD-Encrypt* задается в соответствии со следующей формулой:

$$AEAD\text{-}Encrypt\ (K, nonce, A, P) = C|T, \tag{26}$$

где

$$\begin{gathered}(MGMnonce, A, C, T) = MGMEncrypt\ (K^{seqnum}, MGMnonce, A, P),\\ K^{seqnum} = TLSTREE\ (K, seqnum),\\ MGMnonce = nonce\ [1..1]\&0x7f\,|\,nonce\ [2..IVLen].\end{gathered} \tag{27}$$

Для каждой полученной записи с номером *seqnum* функция расшифрования *AEAD-Decrypt* задается следующим образом.

Входные аргументы:

- $K \in B_{KLen}$, ключ шифрования;
- $nonce \in B_{IVLen}$, уникальный вектор;
- $A \in B_s$, $s \geq 0$, дополнительные имитозащищаемые данные;
- $ENCrecord = C|T$, где $C \in B_{|P|}$ — шифртекст, $T \in B_s$ — имитовставка.

Результат работы:

- *res*, содержащий либо ошибку, либо байтовую строку длины $|C|$.

Функция расшифрования *AEAD-Decrypt* задается в соответствии со следующей формулой:

$$AEAD\text{-}Decrypt\ (K, nonce, A, ENCrecord) = res, \tag{28}$$

где

$$\begin{gathered}res' = MGMDecrypt\ (K^{seqnum}, MGMnonce, A, C, T),\\ C|T = ENCrecord,\\ K^{seqnum} = TLSTREE\ (K, seqnum),\\ MGMnonce\text{-}nonce[1..1]\ \&\ 0x7f|nonce[2..IVLen],\\ res = \begin{cases} P \in B_{|c|}, \text{ в случае если } res' = (A, P);\\ ошибка, \text{ в случае если } res' \text{ содержит ошибку.}\end{cases}\end{gathered} \tag{29}$$

Алгоритм TLSTREE выработки ключей защиты записей определяется в соответствии с 10.1.2.1.

10.1.2.1 Алгоритм TLSTREE выработки ключей защиты записей

В настоящем разделе описывается алгоритм TLSTREE, используемый для порождения ключей защиты записей из корневого ключа $K_{root} \in B_{32}$.

Функция *TLSTREE* задается следующим образом.

Входные аргументы:

- $K_{root} \in B_{32}$, корневой ключ;
- $i \in \{0, 1, \dots, 2^{64} - 1\}$, число.

Результат работы:

- ключевой материал длины 32 байта.

Функция *TLSTREE* задается в соответствии со следующей формулой:

$$TLSTREE\ (K_{root}, i) = Divers_3(Divers_2(Divers_1(K_{root}, STR_8(i \,\&\, C_1)), STR_8(i \,\&\, C_2)), STR_8(i \,\&\, C_3)), \quad (30)$$

где

- $C_1, C_2, C_3 \in \{0, 1, \dots, 2^{64} - 1\}$ — константы, определяемые конкретным криптонабором (см. 10.1);
- $Divers_j\ (K, D)$, $j \in \{1,2,3\}$ — алгоритм диверсификации ключа $K \in B_{32}$ по данным диверсификации $D \in B_8$, который задается с помощью функции KDF_{256}, определяемой алгоритмом KDF_GOSTR3411_2012_256, описанным в Р 50.1.113:

$$\begin{aligned} Divers_1\ (K, D) &= KDF_{256}\ (K, "level1", D); \\ Divers_2\ (K, D) &= KDF_{256}\ (K, "level2", D); \\ Divers_3\ (K, D) &= KDF_{256}\ (K, "level3", D). \end{aligned} \quad (31)$$

10.1.2.2 Параметры ключевого дерева

Константы C_1, C_2, C_3, используемые для порождения ключей защиты записей, определяются в соответствии с таблицей 13.

Таблица 13 — Параметры ключевого дерева

Криптонабор	Константы C_1 C_2, C_3
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_L	0xf800000000000000, 0xffffff000000000, 0xfffffffffffe000
TLS_GOSTR341112_256_WITH_MAGMA_MGM_L	0xffe0000000000000, 0xfffffffc0000000, 0xffffffffffffff80
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_S	0xfffffffe0000000, 0xffffffffffff0000, 0xfffffffffffff8
TLS_GOSTR341112_256_WITH_MAGMA_MGM_S	0xfffffffffc000000, 0xfffffffffffe000, 0xffffffffffffffff

10.1.3 Максимальное количество записей

Параметр *SNMAX* задает максимальное количество записей, которые могут передаваться в рамках одного значения [*sender*]_*write_key* (номер записи *seqnum* может принимать значения от 0 до *SNMAX*-1 включительно), и определяется в соответствии с таблицей 14.

Таблица 14 — Максимальное количество записей

Криптонабор	SNMAX
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_L	2^{64}
TLS_GOSTR341112_256_WITH_MAGMA_MGM_L	2^{64}
TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_S	2^{42}
TLS_GOSTR341112_256_WITH_MAGMA_MGM_S	2^{39}

В случае если номер получаемой/отправляемой записи близок к значению *SNMAX*, стороны могут либо обновить значение ключевого материала трафика с помощью механизма сообщений KeyUpdate (см. 5.9.3), либо завершить соединение.

10.1.4 Хэш-функция

Определенные в настоящих рекомендациях криптонаборы в качестве хэш-функции *HASH* используют хэш-функцию, описанную в ГОСТ Р 34.11, с длиной выхода *HLen* = 32 (256 бит).

10.2 Идентификаторы схем подписи из реестра «TLSSignatureScheme»

Настоящие рекомендации вводят следующие идентификаторы IANA из приватной области реестра «TLSSignatureScheme», указываемые в расширениях signature_algorithms и signature_algorithms_cert:

```
enum {
        gostr34102012_256a(0x0709),
        gostr34102012_256b(0x070A),
        gostr34102012_256c(0x070B),
        gostr34102012_256d(0x070C),
        gostr34102012_512a(0x070D),
        gostr34102012_512b(0x070E),
        gostr34102012_512c(0x070F),
        (0xFFFF)
} SignatureScheme;
```

Каждый из вышеперечисленных идентификаторов из реестра «TLSSignatureScheme» соответствует схеме подписи, определяемой алгоритмом подписи, описанным в ГОСТ Р 34.10, с длиной ключа 256 или 512 бит и одной из эллиптических кривых, описанных в Р 1323565.1.024. Соответствие между вводимыми идентификаторами и схемами подписи приведено в таблице 15.

Таблица 15 — Схемы подписи для идентификаторов из реестра «TLSSignatureScheme»

Наименование схемы подписи	Алгоритм подписи	Идентификатор кривой *E*
gostr34102012_256a	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-tc26-gost-3410-2012-256-paramSetA
gostr34102012_256b	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-tc26-gost-3410-2012-256-paramSetB
gostr34102012_256c	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-tc26-gost-3410-2012-256-paramSetC
gostr34102012_256d	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-tc26-gost-3410-2012-256-paramSetD
gostr34102012_512a	по ГОСТ Р 34.10 с длиной ключа 512 бит	id-tc26-gost-3410-12-512-paramSetA
gostr34102012_512b	по ГОСТ Р 34.10 с длиной ключа 512 бит	id-tc26-gost-3410-12-512-paramSetB
gostr34102012_512c	по ГОСТ Р 34.10 с длиной ключа 512 бит	id-tc26-gost-3410-2012-512-paramSetC

В силу исторических причин помимо идентификаторов кривых, перечисленных в таблице 15, существуют старые значения идентификаторов, которые соответствуют тем же параметрам эллиптических кривых. В целях обеспечения совместимости реализации должны быть готовы поддерживать как новые, так и устаревшие значения идентификаторов (см. таблицу 16).

Таблица 16 — Дополнительные идентификаторы кривых перечисления SignatureScheme

Наименование схемы подписи	Алгоритм подписи	Идентификатор кривой *E*
gostr34102012_256b	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-GostR3410-2001-CryptoPro-A-ParamSet
		id-GostR3410-2001-CryptoPro-XchA-ParamSet
gostr34102012_256c	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-GostR3410-2001-CryptoPro-B-ParamSet
gostr34102012_256d	по ГОСТ Р 34.10 с длиной ключа 256 бит	id-GostR3410-2001-CryptoPro-C-ParamSet
		id-GostR3410-2001-CryptoPro-XchB-ParamSet

При этом функция *SIGN*, используемая для формирования значения подписи в 5.8.2, задается следующим образом.

Входные аргументы:

- $M \in B^*$, произвольная байтовая строка;
- $0 < d_{sign} < q$, ключ подписи, где q — порядок циклической подгруппы группы точек эллиптической кривой E.

Результат работы:

- $sgn \in B_{21}$, где $l \in \{32,64\}$.

Функция *SIGH* задается в соответствии со следующей формулой:

$$(r, s) = SIGNGOST(M, d_{sign}),$$
$$SIGN(M, d_{sign}) = sgn = str_l(r)|str_l(s), \quad (32)$$

где $SIGNGOST(M, d_{sign})$ — алгоритм подписи, выдающий в качестве результата своей работы пару чисел (r, s), выработанных в результате вычисления значения подписи сообщения M на ключе подписи d_{sign} в соответствии с алгоритмом подписи по ГОСТ Р 34.10 с параметрами, соответствующими кривой E;

$l = 32$ для алгоритма подписи *SIGNGOST*, соответствующего алгоритму подписи по ГОСТ Р 34.10 с длиной ключа 256 бит, и $l = 64$ для алгоритма подписи *SIGNGOST*, соответствующего алгоритму подписи по ГОСТ Р 34.10 с длиной ключа 512 бит.

Примечание — В формуле (32) значение подписи *sgn* представляется в виде конкатенации двух строк, являющихся байтовыми представлениями чисел *r* и *s* в формате little-endian.

10.3 Идентификаторы кривых из реестра «TLSSupportedGroups»

Настоящие рекомендации определяют следующие новые идентификаторы IANA из реестра «TLSSupportedGroups», указываемые в расширении supported_groups:

```
enum {
        GC256A(0x0022), GC256B(0x0023), GC256C(0x0024),
        GC256D(0x0025), GC512A(0x0026), GC512B(0x0027),
        GC512C(0x0028),
        (0xFFFF)
} NamedGroup;
```

Каждый из вышеперечисленных идентификаторов определяет одну из эллиптических кривых, описанных в Р 132356.1.024. Соответствие между вводимыми идентификаторами из реестра «TLSSupportedGroups», идентификаторами кривых и длиной координат точек кривой (значение coordinate_length, см. 5.6.4.4) приведено в таблице 17.

Таблица 17 — Задание эллиптических кривых перечисления NamedGroup

Наименование кривой	Идентификатор кривой	Значение coordinate_length
GC256A	id-tc26-gost-3410-2012-256-paramSetA	32
GC256B	id-tc26-gost-3410-2012-256-paramSetB	32
GC256C	id-tc26-gost-3410-2012-256-paramSetC	32
GC256D	id-tc26-gost-3410-2012-256-paramSetD	32
GC512A	id-tc26-gost-3410-12-512-paramSetA	64
GC512B	id-tc26-gost-3410-12-512-paramSetB	64
GC512C	id-tc26-gost-3410-2012-512-paramSetC	64

11 Вопросы реализации и безопасности

11.1 Механизмы защиты от атак по побочным каналам

В целях создания эффективной реализации, а также противодействия атакам по побочным каналам сторонам взаимодействия необходимо придерживаться следующих правил работы:

- при использовании алгоритма TLSTREE обращение к функции $Divers_j$, $j \in \{1,2,3\}$, должно проводиться только в тех случаях, когда номер записи *seqnum* достигает такого значения, что $seqnum \,\&\, C_j \neq (seqnum - 1) \,\&\, C_j$, в противном случае необходимо использовать значение, выработанное ранее;
- для каждого предварительно распределенного секрета *PSK* значение *HMAC_binder_key* должно вычисляться только один раз в рамках всех соединений, в которых тикет, соответствующий данному значению *PSK*, указывался клиентом в расширении pre_shared_key сообщенийClientHello.

11.2 Механизмы защиты от downgrade-атак

11.2.1 Формирование значения поля random в рамках режима совместимости

Данный механизм защищает TLS 1.3 клиента и TLS 1.3 сервер от downgrade-атак и применяется в случае если TLS 1.3 клиент и сервер допускают работу в рамках режима совместимости.

При работе TLS 1.3 сервера в режиме совместимости в случае если максимально поддерживаемая версия протокола TLS, указанная клиентом в сообщении ClientHello, соответствует версии 1.2 и ниже, сервер должен установить последние 8 байт значения поля random в сообщении ServerHello равными следующим специальным значениям:

- при выборе протокола TLS 1.2: 44 4F 57 4E 47 52 44 01;
- при выборе протокола TLS 1.1 или ниже: 44 4F 57 4E 47 52 44 00.

При работе TLS 1.3 клиента в режиме совместимости при получении сообщения ServerHello, соответствующего версии протокола TLS 1.2 и ниже, клиент должен проверить, что последние 8 байт поля ServerHello.random не равны ни одному из перечисленных выше значений. В противном случае клиент должен прекратить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

Примечание — TLS 1.2 клиенту, получившему сообщение ServerHello, соответствующее версии протокола TLS 1.1 и ниже, рекомендуется также проверить, что последние 8 байт поля ServerHello.random не равны второму из перечисленных выше значений. В противном случае клиент должен прекратить работу протокола Handshake с оповещением illegal_parameter (см. 7.2).

11.2.2 Использование значения TLS_FALLBACK_SCSV в рамках режима совместимости

Данный механизм защищает от downgrade-атак TLS 1.3 клиента, устанавливающего соединение в рамках режима совместимости с сервером версии 1.2 и ниже, а также TLS 1.3 сервер, устанавливающий соединение в рамках режима совместимости с клиентом версии 1.2 и ниже.

Значение TLS_FALLBACK_SCSV является сигнальным криптонабором, указываемым клиентом в поле cipher_suites сообщения ClientHello после всех значений поддерживаемых криптонаборов и определяется в соответствии с [3] следующим образом:

CipherSuite TLS_FALLBACK_SCSV = {0x56, 0x00};

Данное значение не используется для согласования криптографических алгоритмов, не может быть выбрано сервером в рамках работы протокола Handshake и используется для информирования сервера о том, что клиент пытается повторно установить соединение с понижением версии протокола.

Клиенту рекомендуется указывать значение TLS_FALLBACK_SCSV в поле cipher_suites сообщения ClientHello, если клиент указывает версию в поле legacy_version меньше максимальной версии, поддерживаемой им. В случае если клиент указывает максимальную поддерживаемую версию в поле legacy_version, значение TLS_FALLBACK_SCSV указываться не должно.

При возобновлении соединения клиент не должен указывать значение TLS_FALLBACK_SCSV в поле cipher_suites сообщения ClientHello, поскольку предполагается, что к этому моменту клиент уже знает максимальную версию протокола TLS, поддерживаемую сервером.

При получении сообщения ClientHello, содержащего в поле cipher_suites значение TLS_FALLBACK_SCSV, сервер действует следующим образом:

- в случае если максимальная версия протокола, поддерживаемая сервером, выше версии, указанной клиентом в сообщении ClientHello в поле legacy_version, сервер должен завершить работу протокола Handshake с оповещением inappropriate_fallback. При этом значение поля legacy_record_version в незащищенной записи, содержащей указанное оповещение, должно быть равным либо значению поля ClientHello.legacy_version, либо значению поля legacy_record_version в незащищенной записи, содержащей сообщение ClientHello;
- в случае если значение TLS_FALLBACK_SCSV не было указано или при указании данного значения максимальная версия протокола, поддерживаемая сервером, не превосходит версию протокола, указанную клиентом в поле ClientHello.legacy_version) сервер продолжает работу протокола Handshake в штатном режиме.

Приложение А
(справочное)

Рекомендации по использованию TLS 1.3 криптонаборов в СКЗИ

В настоящем приложении приводятся рекомендации по использованию описанных криптонаборов в СКЗИ в зависимости от области их применения.

Регулируемые государством области применения СКЗИ определяются в соответствии с [4]. Для СКЗИ, применяемых в данных областях, вводится классификация, определенная в Р 1323565.1.012. Далее приводятся рекомендации по использованию описанных в настоящем документе криптонаборов в СКЗИ в соответствии с данной классификацией.

В СКЗИ, относящихся к классам КС1, КС2, КС3, допустимо использовать все криптонаборы, определяемые в настоящем документе.

В СКЗИ, относящихся к классу КВ, рекомендуется использовать следующие криптонаборы, определяемые в настоящем документе:

- TLS_GOSTR341112_256_WITH_KUZNYECHIK_MGM_S;
- TLS_GOSTR341112_256_WITH_MAGMA_MGM_S.

При использовании криптонабора TLS_GOSTR341112_256_WITH_MAGMA_MGM_S рекомендуется ограничивать размер данных, используемых при формировании записей в протоколе Record, так, чтобы размер поля TLSPlaintext.fragment не превышал 1 КБ (значение поля TLSPlaintext.length, соответственно, не будет превышать 1024).

Приложение Б
(справочное)

Язык представления данных в протоколе TLS

В настоящем приложении содержится описание общепринятого языка представления данных в протоколе TLS 1.3.

Б.1 Размер базового блока данных

Представление всех элементов данных указывается в явном виде. Размер базового блока данных, передаваемых в рамках работы протокола TLS 1.3, составляет 1 байт (8 бит). Многобайтовые элементы данных являются конкатенацией (объединением) байт слева направо, сверху вниз и представляются в виде байтовых строк в формате big-endian.

Б.2 Разное

Текст комментария начинается с символов «/*» и заканчивается символами «*/».

Необязательные (опциональные) компоненты выделяются с помощью двойных квадратных скобок «[[ ]]».

Тип элементов размером в 1 байт, содержащих не интерпретируемые в рамках работы протокола TLS данные, обозначается типом opaque.

Переобозначение T′ (type alias) для существующего типа T задается следующим образом:

```
TT′;
```

Б.3 Числа

Базовым типом числовых данных является беззнаковый байт (uint8). В настоящих рекомендациях используются следующие предопределенные типы числовых данных:

```
uint8 uint16[2];
uint8 uint24[3];
uint8 uint32[4];
uint8 uint64[8];
```

Все числовые значения указанных типов представляются в виде байтовых строк в формате big-endian.

Б.4 Векторы

Вектор (одномерный массив) представляет собой поток элементов данных одного и того же типа. Длина вектора задается в байтах и может быть указана во время объявления или оставаться неопределенной вплоть до начала работы протокола TLS.

Вектор T′ фиксированной длины, содержащий данные типа T, задается следующим образом:

```
TT′[n];
```

где значение n является длиной вектора T′ в байтах и кратно размеру T. При этом длина вектора не включается в кодированный поток данных.

Байтовое представление значения, являющегося вектором, задается конкатенацией байтовых представлений элементов данного вектора в порядке их нумерации (слева направо).

В приведенном ниже примере вектор Datum определяется как три последовательных байта, не интерпретируемых протоколом, в то же время вектор Data определяется как три последовательных вектора Datum, состоящие в общей сложности из 9 байт.

```
opaque Datum[3];    /* три неинтерпретируемых байта */
Datum Data[9];      /*3 последовательных 3-байтовых вектора */
```

Векторы переменной длины определяются с указанием допустимого поддиапазона размеров (включая крайние значения) в формате <floor..ceiling>. При кодировании в поток данных перед содержимым вектора помещается его фактическая длина. Значение длины данного вектора представляется в виде байтовой строки с длиной, равной длине строки, требуемой для хранения максимального значения длины вектора (ceiling). Вектор переменной длины, имеющий фактическую нулевую длину, представляется в виде байтовой строки, соответствующей нулевому значению.

Вектор T′ переменной длины, содержащий данные типа T, задается следующим образом:

```
TT′<floor..ceiling>;
```

При этом длина кодированного вектора должна быть в точности кратна размеру одиночного элемента (например, 17-байтовый вектор типа uint16, будет недопустимым).

В следующем примере первый вектор mandatory имеет тип opaque и размер от 300 до 400 байт (такой вектор никогда не может быть пустым). Поле фактической длины вектора занимает 2 байта (uint16), которых достаточно для записи максимальной длины вектора, равной 400. Второй вектор longer может содержать до 800 байт данных или до 400 элементов uint16 и может быть вектором нулевой длины. Его кодирование будет включать поле размером 2 байта, соответствующее длине вектора и предшествующее его элементам.

```
opaque mandatory<300..400>;
        /* поле длины занимает 2 байта, не может быть пустым */
uint16  longer<0..800>;
        /* от 0 до 400 16-битовых целых чисел без знака */
```

Б.5 Перечисления enum

В рамках работы протокола TLS 1.3 используется дополнительный тип разреженных данных — перечисление enum. Каждое определение перечисления задает новый тип. Операции присваивания и сравнения могут использовать только элементы перечисления одного и того же типа.

Перечисление задается следующим образом:

```
enum { e1(v1), e2(v2), ... , en(vn) [[, (n)]] } Te;
```

где

- v1,v2,...,vn — значения элементов e1,e2,...,en соответственно;
- (n) — опциональный элемент, содержащий максимальное возможное значение, которое может принимать элемент данного перечисления, и предназначенный для определения байтового размера каждого элемента перечисления.

Поскольку элементы перечисления не упорядочены, им может быть присвоено любое уникальное значение в любом порядке.

Будущие расширения или дополнения к протоколу TLS 1.3 могут определять новые значения элементов перечислений. Реализации должны иметь возможность анализировать и игнорировать неизвестные значения, если в определении поля перечисления не указано иное.

Значение каждого элемента перечисления может быть представлено в виде байтовой строки, равной по длине байтовой строке, соответствующей наибольшему значению среди всех значений элементов данного перечисления. В следующем примере элементы перечисления Color будут занимать в потоке по 1 байту.

```
enum { red(3), blue(5), white(7) } Color;
```

Для того, чтобы задать байтовый размер элементов перечисления без определения лишнего элемента, можно дополнительно задать значение без сопоставления с ним соответствующего наименования. В следующем примере элемент перечисления Taste будет занимать 2 байта в потоке данных, но в текущей версии протокола может принимать значения только 1, 2 или 4.

```
enum { sweet(1), sour(2), bitter(4), (32000) } Taste;
```

Наименования элементов перечисления допустимы в пределах заданного типа. В первом примере полностью корректное обращение ко второму элементу перечисления будет иметь вид Color.blue. Подобное уточнение не требуется, если цель присвоения точно задана.

```
Color color = Color.blue; /* переопределение, допустимо */
Color color = blue;       /* корректно, тип задан неявно */
```

Наименования, присвоенные элементам перечисления, не обязательно должны быть уникальными. Одному и тому же элементу перечисления может соответствовать значение, обозначающее числовой диапазон. Данное значение включает в себя минимальное и максимальное значения, содержащиеся в указанном диапазоне, которые разделяются двумя подряд идущими точками, как показано в следующем примере.

```
enum { sad(0), meh(1..254), happy(255) } Mood;
```

Описанное представление элементов перечисления может быть использовано преимущественно для резервирования областей в памяти.

Б.6 Структуры

Структуры представляют собой типы данных, которые могут быть сформированы из ранее определенных типов данных. Каждое определение структуры задает новый уникальный тип.

Структура задается следующим образом:

```
struct {
    T1 f1;
    T2 f2;
    ...
    Tnfn;
} T;
```

где f1, f2, …, fn являются полями структуры T, значения которых имеют типы T1, T2, …, Tn соответственно.

Байтовое представление данных, соответствующих определенной структуре, задается конкатенацией байтовых представлений значений полей структуры в порядке их объявления (сверху вниз).

Векторы допустимы в качестве полей структуры, имеющих фиксированную или переменную длину, и формируются в соответствии с разделом Б.4. Примерами структур, содержащих вектор в качестве своего поля, являются структуры V1 и V2, описанные в Б.8.

Обращение к полям внутри структуры может быть осуществлено с использованием наименований элементов с применением такого же синтаксиса, который описан для перечислений. Например, обращение T.f2 будет указывать на второе поле определенной выше структуры T. Определения структур могут быть вложенными.

Б.7 Константы

Полям структур и переменным могут быть присвоены фиксированные значения с помощью оператора «=», как в следующем примере:

```
struct {
    T1 f1 = 8; /* T.f1 всегда должно быть равно 8 */
    T2 f2;
} T;
```

Б.8 Оператор выбора select

Определяемые структуры могут содержать варианты формирования, выбор между которыми производится при помощи оператора select и основывается на доступной в среде информации. Переключатель (селектор) вариантов должен быть перечислением (см. Б.5), которое определяет возможные варианты формирования структуры данных и задается следующим образом:

```
enum { e1(v1), e2(v2), …, en(vn) [[, (n)]] } E;

    struct {
    select (E) {
        case e1:  Te1  [[fe1]];
        case e2:  Te2  [[fe2]];
        ....
        case en:  Ten  [[fen]];
    };
} Tv;
```

Каждая ветка оператора select определяет тип поля данного варианта и необязательное наименование поля. Механизм выбора варианта во время работы протокола TLS не описывается данным языком представления.

Ниже приведен следующий пример формирования структуры на основе использования оператора select.

```
enum { apple(0), orange(1) } VariantTag;

struct {
    uint16 number;
    opaque string<0..10>; /* вектор переменной длины */
} V1;

struct {
    uint32 number;
    opaque string[10]; /* вектор фиксированной длины */
} V2;

struct {
    VariantTag type;
    select (VariantRecord.type) {
        case apple: V1;
        case orange: V2;
    };
} VariantRecord;
```

Библиография

[1] IETF RFC 8446 E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, IETF RFC 8446

[2] IETF RFC 5705 E. Rescorla, Keying Material Exporters for Transport Layer Security (TLS), IETF RFC 5705

[3] IETF RFC 7507 B. Moeller, A. Langley, TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks, IETF RFC 7507

[4] Приказ ФСБ России от 9 февраля 2005 г. № 66 (в редакции приказа ФСБ России от 12 апреля 2010 г. № 173). Об утверждении положения о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ—2005)

УДК 681.3.06:006.354 ОКС 35. 040 ОКСТУ 5002

Ключевые слова: криптографические протоколы, аутентификация, пароль, ключ

БЗ 4—2020/1

Редактор *Е.А. Моисеева*
Технический редактор *И.Е. Черепкова*
Корректор *М.В. Бучная*
Компьютерная верстка *Е.А. Кондрашовой*

Сдано в набор 28.02.2020. Подписано в печать 01.06.2020. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 8,37. Уч.-изд. л. 7,53.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении во ФГУП «СТАНДАРТИНФОРМ»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.

gost.e-ecolog.ru gost.e-ecolog.ru

gost.e-ecolog.ru