Internet Engineering Task Force (IETF)
Request for Comments: 9147
Obsoletes: 6347
Category: Standards Track
ISSN: 2070-1721

E. Rescorla
Mozilla
H. Tschofenig
Arm Limited
N. Modadugu
Google, Inc.

April 2022

The Datagram Transport Layer Security (DTLS) Protocol Version 1.3

Протокол защиты транспорта дейтаграмм (DTLS) версии 1.3

Аннотация

Этот документ задаёт версию 1.3 протокола защиты транспорта дейтаграмм (Datagram Transport Layer Security или DTLS). DTLS 1.3 позволяет приложениям клиентов и серверов взаимодействовать через Internet без возможности просмотра, перехвата и подмены сообщений.

Протокол DTLS 1.3 основан на протоколе защиты транспортного уровня (Transport Layer Security или TLS) версии 1.3 и обеспечивает такие же гарантии безопасности за исключением поддержки порядка доставки и невоспроизводимости пакетов. Семантика дейтаграмм базового транспорта сохраняется в DTLS.

Этот документ отменяет RFC 6347.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информация о текущем статусе документа, найденных ошибках и способах обратной связи доступна по ссылке https://www.rfc-editor.org/info/rfc9147.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Введение

Основной целью протокола TLS является организация аутентифицированных соединений между взаимодействующими партнёрами и обеспечение для соединений защиты целостности и конфиденциальности. Протокол TLS включает два уровня - протокол TLS record и протокол TLS handshake. Для работы TLS требуется надёжный транспортный канал, обычно TCP [RFC0793].

Некоторые приложения используют транспорт UDP [RFC0768] и протокол DTLS был разработан для обеспечения защиты взаимодействий таких приложений. DTLS намеренно сделан максимально похожим на TLS, чтобы свести к минимуму новые разработки для защиты и максимально воспользоваться имеющимся кодом и инфраструктурой.

DTLS 1.0 [RFC4347] был разработан на основе TLS 1.1 [RFC4346], DTLS 1.2 [RFC6347] - на основе TLS 1.2 [RFC5246]. Протокола DTLS 1.1 нет, эта версия была пропущена для согласования с нумерацией версий TLS. Эта спецификация описывает наиболее современную версию протокола DTLS, основанную на TLS 1.3 [TLS13] и отменяет DTLS 1.2.

Реализации, поддерживающие DTLS 1.2 и DTLS 1.3, функционально совместимы с поддерживающими лишь DTLS 1.2 (используется DTLS 1.2), а реализации TLS 1.3 совместимы с TLS 1.2 (см. Приложение D к [TLS13]). Хотя возможна совместимость и с DTLS 1.0, применять DTLS 1.0 не рекомендуется, как указано в параграфе 3.1.2 [RFC7525]. Документ [DEPRECATE] запрещает использовать DTLS 1.0.

2. Соглашения и термины

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

Ниже даны определения используемых в документе терминов.

client - клиент
Конечная точка, инициирующая соединение DTLS.

association - ассоциация
Общее состояние двух конечных точек, организованное при согласовании DTLS.

connection - соединение
Синоним для ассоциации.

endpoint - конечная точка
Клиент или сервер в соединении.

epoch - эпоха
Один набор криптографических ключей, применяемых для шифрования и расшифровки.

handshake - согласование (приветствие)
Начальное согласование между клиентом и сервером, устанавливающее параметры соединения.

peer - партнёр
Конечная точка. При обсуждении конкретной конечной точки партнёром называется другая сторона соединения.

receiver - получатель
Конечная точка, принимающая записи.

sender - отправитель
Конечная точка, передающая записи.

server - сервер
Конечная точка, которая воспринимает (не инициирует) соединение DTLS.

CID
Идентификатор соединения.

MSL
Максимальный срок действия (жизни) сегмента.

Предполагается знакомство читателя с [TLS13]. Как и в TLS 1.3, сообщение HelloRetryRequest имеет такой же формат как ServerHello, но для удобства в документе используется термин HelloRetryRequest, как будто это разные сообщения.

В DTLS 1.3 применяется сетевой порядок байтов (big-endian - сначала старший) кодирования сообщений, заданный в [TLS13] и ранних спецификациях (D)TLS.

Предполагается знакомство читателя с [RFC9146], поскольку функциональность CID применяется в DTLS 1.3.

На рисунках в этом документе приводятся различные варианты обмена сообщениями DTLS с использованием показанных ниже комбинаций символов.

+
Заслуживающие внимания расширения переданы в предыдущем сообщении.

*****
Необязательные сообщения или расширения, передаваемые в зависимости от ситуации.

{}
Сообщения, защищённые с использованием ключей, выведенных из [sender]_handshake_traffic_secret.

[]
Сообщения, защищённые с использованием ключей, выведенных из traffic_secret_N.

3. Обоснование решения и обзор DTLS

Основой решения DTLS является организация TLS через транспортный уровень на базе дейтаграмм. Такой уровень не требует и не предоставляет надёжной и упорядоченной доставки данных. Протокол DTLS сохраняет эти свойства для данных приложений. Приложения вроде потоковой передачи, Internet-телефонии, сетевых игр используют транспорт на основе дейтаграмм из-за чувствительности передаваемых данных к задержкам в сети. Поведение таких приложений не меняется в результате применения протокола DTLS для защиты коммуникаций, поскольку DTLS не компенсирует потери и нарушение порядка доставки. Отметим, что в потоковых приложениях с малой задержкой и играх DTLS применяется для защиты данных (например, канала данных WebRTC), а в телефониии DTLS служит для организации ключей, тогда как защиту данных обеспечивает протокол SRTP[1] [RFC5763].

TLS не подходит напрямую для транспорта дейтаграмм по 4 указанным ниже причинам.

1. TLS полагается на неявные порядковые номера записей и если запись не пришла, получатель будет использовать неверный порядковый номер при попытке снять защиту следующей записи. В DTLS записи содержат явный порядковый номер.
2. Согласование TLS - это пошаговый криптографический протокол, гда сообщения должны передаваться и приниматься в заданном порядке, нарушение которого является ошибкой. Согласование DTLS включает порядковые номера сообщений, позволяющие собрать фрагменты и восстановить порядок сообщений.
3. Сообщения при согласовании могут превышать доступный для дейтаграмм размер. DTLS добавляет в согласующие сообщения поля, позволяющие собирать фрагменты.
4. Протоколы доставки дейтаграмм подвержены некорректному поведени, вызывающему атаки с отказом в обслуживании (denial-of-service или DoS) сторонних элементов. В DTLS 1.3 применяется проверка обратных маршрутов с помощью сообщений TLS 1.3 HelloRetryRequest (смю 5.1. Противодействие DoS-атакам).

3.1. Потеря пакетов

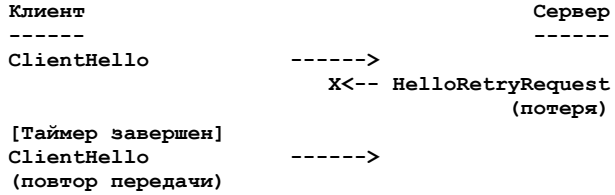

Рисунок 1. Пример повтора передачи DTLS.

[1]Secure Real-time Transport Protocol - защищенный протокол транспортировки в реальном масштабе времени.

В DTLS применяется простой таймер повтора передачи для обработки случаев потери пакетов.

На рисунке 1 показана базовая концепция использования первой фазы согласования DTLS (handshake).

Как только клиент передал сообщение ClientHello, он ожидает от сервера HelloRetryRequest или ServerHello. Однако по завершении отсчёта запущенного таймера клиент понимает, что ClientHello или отклик от сервера потерян и повторяет отправку ClientHello. Когда сервер получает повтор, он понимает, что нужно повторить своё сообщение HelloRetryRequest или ServerHello.

Сервер тоже поддерживает таймер повтора передачи для отправляемых сообщений (отличных от HelloRetryRequest). Отказ от повтора HelloRetryRequest по таймеру позволяет избежать создания на сервере ненужного состояния. Сообщение HelloRetryRequest достаточно и не будет фрагментироваться, что позволяет избежать проблем с чередованием нескольких HelloRetryRequest.

Более подробное описание тайм-аутов и повтора передачи приведено в параграфе 5.8. Тайм-ауты и повтор передачи.

3.2. Изменение порядка доставки

В DTLS каждое сообщение согласования имеет порядковый номер и получатель может быстро узнать, является ли это сообщение следующим ожидаемым. Если это не так, сообщение помещается в очередь для обработки после приёма его предшественников.

3.3. Фрагментация

Согласующие сообщения TLS и DTLS могут быть достаточно большими (в теории до 2^{24}-1 байтов, практически много кбайт). Дейтаграммы UDP часто ограничены размером 1500 байтов, если фрагментация IP нежелательна. Чтобы преодолеть это ограничение, согласующие сообщения DTLS можно фрагментировать на записи DTLS, каждую из которых можно поместить в одну дейтаграмму UDP (см. параграф 4.4. Проблемы PMTU). В каждом согласующем сообщении DTLS указывается смещение и размер фрагмента, что позволяет получателю собрать исходное сообщение по получении всех фрагментов.

3.4. Обнаружение повторного использования

DTLS может поддерживать обнаружение воспроизведения записей (replay). Используются те же методы, что и в IPsec AH/ESP, путём поддержки окна юитовых отображений принятых пакетов. Записи, не попадающие в окно и полученные ранее автоматически отбрасываются без уведомления. Функция защиты от повтороного использования является необязательной, поскольку дубликаты пакетов не всегда являются злонамеренными и могут возникать из-за ошибок маршрутизации. Приложения могут (предположительно) сами обнаруживать дубликаты пакетов и соответственно изменять свою стратегию передачи данных.

4. Уровень DTLS Record

Уровень записей DTLS 1.3 отличается от уровня записей TLS 1.3 и уровня записей DTLS 1.2.

1. В структуре DTLSCiphertext нет лишних полей номера версии и типа.
2. В DTLS добавлена эпоха и порядковый номер в заголовок записи TLS. Номер позволяет получателю корректно расшифровать и проверить запись DTLS. Число битов для полей эпохи и номера в структуре DTLSCiphertext сокращено по сравнению с предыдущей версией.
3. Эпоха DTLS представленная в DTLSPlaintext имеет размер 2 октета для совместимости с DTLS 1.2. Это значение помещается в два младших октета эпохи соединения, заданной 8-октетным счётчиком, инкрементируемым при каждом обновлении ключей KeyUpdate (4.2. Порядковый номер и эпоха). Порядковый номер помещается в младшие 48 битов 64-битового поля номера. Нешифрованные записи **недопустимо** передавать с порядковыми номерами больше 2^{48}-1, поэтому 16 старших битов всегда имеют значение 0.
4. Заголовок структуры DTLSCiphertext имеет переменный размер.

Структуры DTLSPlaintext служат для передачи незащищённых записей, а DTLSCiphertext - для передачи защищенных.

Форматы записей DTLS показаны на рисунке 2. Если явно не указано иное, поля имеют такое же значение, как в предыдущих версиях TLS/DTLS.

```
struct {
    ContentType type;
    ProtocolVersion legacy_record_version;
    uint16 epoch = 0
    uint48 sequence_number;
    uint16 length;
    opaque fragment[DTLSPlaintext.length];
} DTLSPlaintext;

struct {
     opaque content[DTLSPlaintext.length];
     ContentType type;
     uint8 zeros[length_of_padding];
} DTLSInnerPlaintext;

struct {
    opaque unified_hdr[variable];
    opaque encrypted_record[length];
} DTLSCiphertext;
```

Рисунок 2. Форматы записей DTLS 1.3.

legacy_record_version

Поле должно иметь значение {254, 253} для всех записей, кроме начальной ClientHello (т. е. не созданной после HelloRetryRequest), где оно может иметь значение {254, 255} для совместимости. Поле должно игнорироваться при обработке (см. Приложение D.1 к [TLS13]).

epoch

Младшие 2 байта значения эпохи соединения.

unified_hdr

Структура переменного размера, показанная на рисунке 3.

encrypted_record

Зашифрованная форма последовательного представления структуры DTLSInnerPlaintext.

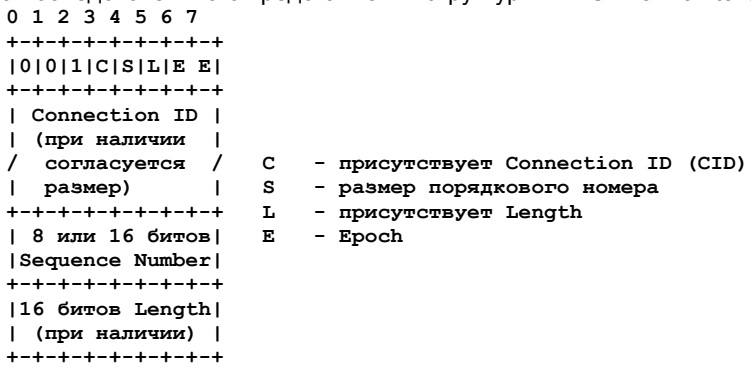

Рисунок 3. Унифицированный заголовок DTLS 1.3.

Фиксированные биты

Три старших бита унифицированного заголовка имеют значения 001. Это гарантирует соответствие значения области DTLS при мультиплексировании в соответствии с [RFC7983], а также позволяет гарантированно отличать шифрованные записи DTLS 1.3 от шифрованных записей DTLS 1.2 при их передаче с одним квартетом хостов и портов. Такое мультиплексирование возможно лишь при использовании CID [RFC9146], где записи DTLS 1.2 будут иметь тип содержимого tls12_cid (25).

C

Бит (0x10), устанавливаемый при наличии Connection ID.

S

Этот бит (0x08) указывает размер порядкового номера - 0 для 8-битового, 1 для 16-битового. Реализации **могут** применять в одном соединении номера разного размера.

L

Бит L (0x04) указывает наличие поля Length.

E

Указывают 2 младших бита эпохи.

Connection ID

Идентификатор соединения (CID) переменного размера. Функциональность CID описана в [RFC9146], а пример можно найти в параграфе 9.1. Пример CID.

Sequence Number

8 или 16 битов порядкового номера записи (16 при установленном флаге S, 8 при сброшенном).

Length

Идентично полю Length в записи TLS 1.3.

Как и в прежних версиях DTLS, можно включить несколько записей DTLSPlaintext и DTLSCiphertext в одну дейтаграмму базового транспорта.

На рисунке 4 показаны разные заголовки записей.

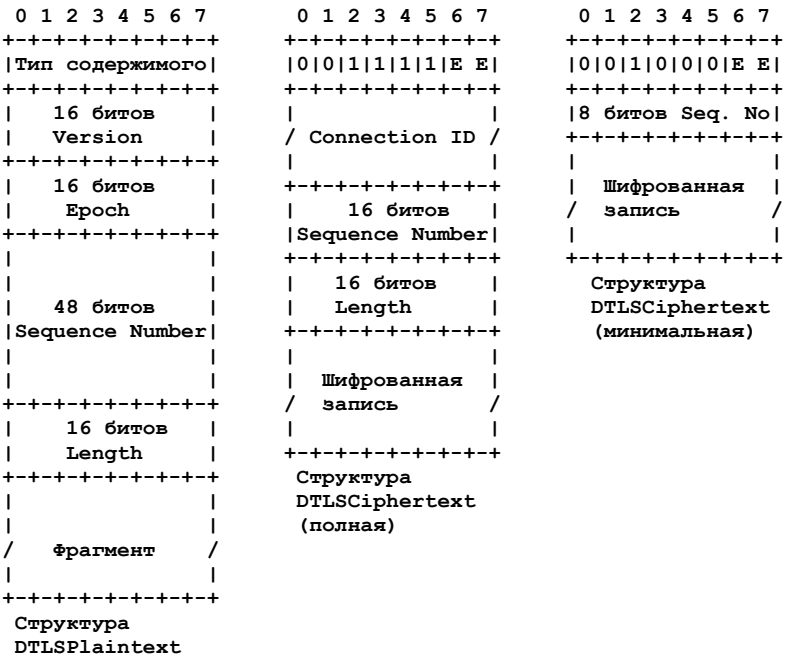

Рисунок 4. Примеры заголовков DTLS 1.3.

Поле Length **можно** опустить, сбросив флаг L, это будет указывать, что запись занимает все пространство до конца транспортной дейтаграммы. Невозможно иметь внутри одной дейтаграммы несколько записей DTLSCiphertext без поля размера. Поле размера **можно** не указывать лишь в последней записи дейтаграммы, все остальные записи **должны** включать это поле. Реализации **могут** смешивать в одном соединении записи с полем размера и без него.

Если значение Connection ID согласовано, оно **должно** включаться во все дейтаграммы. Передающей реализации **недопустимо** смешивать в одной дейтаграмме записи разных ассоциаций DTLS. Если во второй или последующих записях указан идентификатор соединений, не совпадающий с использованным в предыдущих записях той же ассоциации, остальная часть дейтаграммы **должна** отбрасываться.

После извлечения эпоху и порядковый номер можно объединить в структуре RecordNumber

```
struct {
    uint64 epoch;
    uint64 sequence_number;
} RecordNumber;
```

Это 128-битовое значение применяется в сообщении ACK, а также в качестве входного параметра record_sequence_number функции аутентифицированного шифрования со связанными данными (Authenticated Encryption with Associated Data или AEAD). Значение заголовка целиком, показанное на рисунке 4 (до шифрования номера записи, см. 4.2.2. Восстановление порядкового номера и эпохи), применяется как дополнительные данные функции AEAD. Например, при использовании минимального варианта связанные данные (Associated Data или AD) имеют размер 2 октета. Отметим, что это отличается от расчёта дополнительных данных для DTLS 1.2 и DTLS 1.2 с Connection ID. В DTLS 1.3 64-битовое значение sequence_number используется как порядковый номер для расчёта AEAD и, в отличие от DTLS 1.2, эпоха не включается.

4.1. Демультиплексирование записей DTLS

Формат заголовка DTLS 1.3 сложнее, чем в DTLS 1.2, где первый байт всегда указывает тип содержимого. Как показано на рисунке 5, первый байт определяет способ демультиплексирования входящей записи DTLS. Первые 3 бита отличают шифрованные записи DTLS 1.3 от записей прежних версий и нешифрованных записей DTLS 1.3. Поэтому диапазон от 32 (0b0010 0000) до 63 (0b0011 1111) должен быть исключён в будущем из выделяемых IANA значений, чтобы не было проблем при демультиплексировании (см. раздел 14). Реализации могут демультиплексировать записи DTLS 1.3, проверяя первый байт, как описано ниже.

- Если первый байт - это alert(21), handshake(22) или ack(proposed, 26), запись **должна** считаться DTLSPlaintext.
- В иных случаях получатель **должен** проверить, имеют ли первые биты значение 001. Если так, реализация **должна** обработать запись как DTLSCiphertext, фактический тип содержимого будет в защищённой части.
- В противном случае запись **должна** отвергаться как при отказе снятия защиты (см. параграф 4.5.2).

На рисунке 5 демультиплексирование представлено с учётом DTLS 1.3 и более ранних версий DTLS.

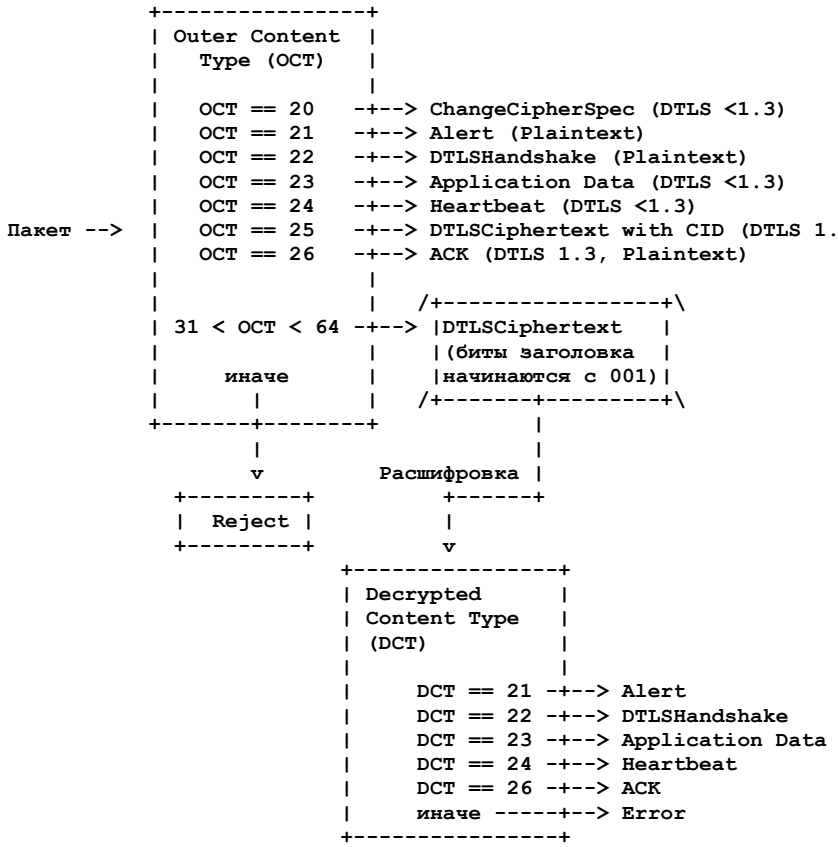

Рисунок 5. Демультиплексирование записей DTLS 1.2 и DTLS 1.3.

4.2. Порядковый номер и эпоха

DTLS использует явный или частично явный порядковый номер вместо неявного, передавая его в поле записи sequence_number. Порядковые номера поддерживаются раздельно для каждой эпохи, начиная с sequence_number=0.

Для номера эпохи исходно устанавливается значение 0, а затем номер инкрементируется при каждом обновлении ключевого материала, когда отправитель хочеть заменить ключи (см. 6.1. Значения эпохи и смена ключей).

4.2.1. Рекомендации по обработке

Поскольку порядок записей DTLS может быть изменен, запись из эпохи M может быть получена после начала эпохи N (N > M). Реализациям **следует** отбрасывать записи из более ранних эпох, но **можно** сохранять ключевой материал предыдущих эпох в течение принятого по умолчанию интервала MSL для TCP [RFC0793], чтобы принять пакеты с нарушенным порядком. Отметим, что намерение здесь заключается в использовании текущих рекомендаций IETF для MSL, как указано в [RFC0793] или его преемниках, а не в попытке получить значение MSL, применяемое стеком TCP.

Записи, защищённые с новой эпохой, могут приходить до завершения согласования. Например, сервер может передать своё сообщение Finished и после этого начать передачу данных. Реализации **могут** буферизовать или отбрасывать такие записи, хотя при использовании DTLS с надёжным транспортом (например, SCTP [RFC4960]), им **следует** буферизовать записи и обрабатывать их по завершении согласования. Отметим, что ограничения TLS для времени отправки записей сохраняются здесь и получатель обрабатывает такие записи как при получении в корректном порядке.

Реализации **должны** повторять передачу потерянных сообщений с использованием эпохи и ключевого материала, которые применялись для исходной передачи.

Реализация **должна** разорвать ассоциацию или поменять ключи до достижения максимального порядкового номера.

Реализациям **недопустимо** переносить (wrap) эпоху, вместо этого **должна** создаваться новая ассоциация с разрывом прежней.

4.2.2. Восстановление порядкового номера и эпохи

У получателя защищённых пакетов DTLS нет полного значения эпохи и порядкового номера в записи, поэтому существует некоторая вероятность неоднозначности. Поскольку полный порядковый номер применяется при расчёте nonce для записи, а эпоха определяет ключи, невозможность восстановить эти значения ведёт к отказу при снятии защиты записи. Поэтому реализации могут по своему усмотрению применять механизм восстановления полных значений. В этом параграфе представлен сравнительно простой механизм, который **рекомендуется** для реализаций.

Если биты эпохи совпадают с битами текущей эпохи, реализации **следует** восстанавливать порядковый номер путём расчёта полного порядкового номера, максимально близкого численно к имеющемуся номеру и наибольшему номеру, для которого удалось снять защиту записи в текущей эпохе.

На этапе согласования биты эпохи однозначно указывают ключ для использования. По завершении согласования, если биты эпохи не совпадают с битами текущей эпохи, реализации **следует** использовать наиболее близкую прошлую эпоху, соответствующую битам, а затем восстановить порядковый номер для этой эпохи, как описано выше.

4.2.3. Шифрование номера записи

В DTLS 1.3 при шифровании записи шифруется и её порядковый номер. Базовый алгоритм шифрования, используемый AEAD, служит для создания маски, которая применяется в операции XOR с порядковым номером. Когда в AEAD применяется алгоритм AES, маска создаётся путём расчёта AES-ECB для первых 16 байтов шифроданных

```
Mask = AES-ECB(sn_key, Ciphertext[0..15])
```

При использовании в AEAD алгоритма ChaCha20 маска создаётся путём использования первых 4 байтов шифроданных как счётчика блоков, а следующих 12 байтов - как nonce с передачей блочной функции ChaCha20 (см. параграф 2.3 в [CHACHA])

```
Mask = ChaCha20(sn_key, Ciphertext[0..3], Ciphertext[4..15])
```

Значение sn_key определяется выражением

```
[sender]_sn_key = HKDF-Expand-Label(Secret, "sn", "", key_length)
```

где [sender] указывает передающую сторону. Значение Secret для каждой эпохи применяется в соответствии с параграфом Section 7.3 в [TLS13]. Отметим, что для каждой эпохи применяется новый ключ, а поскольку epoch передаётся в открытом виде, неоднозначности не возникает.

Зашифрованный порядковый номер рассчитывается с помощью операции XOR для начальных байтов маски и представляемого в линию порядкового номера. Расшифровка выполняется таким же способом.

Для этой процедуры требуется, чтобы размер шифроданных был не менее 16 байтов. Получатель **должен** отвергать более короткие записи, как будто не удалось снять защиту (4.5.2. Обработка недействительных записей). Отправитель **должен** дополнять короткие открытые данные (используя обычный механизм заполнения), чтобы обеспечить подходящий размер шифроданных. Отметим что большинство влгоритмов AEAD в DTLS имеет 16-байтовый тег аутентификации и не требует дополнения, однако некоторые алгоритмы (например, TLS_AES_128_CCM_8_SHA256) имеют более короткий тег аутентификации и могут требовать заполнения при кратких данных на входе.

Будущие шифронаборы, не основанные на AES или ChaCha20, **должны** задавать своё шифрование порядковых номеров при использовании с DTLS.

Отметим, что шифрование порядковых номеров применяется лишь к структуре DTLSCiphertext, а номера в DTLSPlaintext не шифруются.

4.3. Отображение транспортного уровня

Сообщения DTLS **могут** фрагментироваться на несколько записей DTLS. Каждая запись DTLS **должна** помещаться в одну дейтаграмму. Для предотвращения фрагментации IP клиентам уровня записи DTLS **следует** пытаться устанавливать размер записей так, чтобы они помещались в пакет размером Path MTU (PMTU), оценка которого получена от уровня записи (см. 4.4. Проблемы PMTU).

В одну дейтаграмму **можно** поместить несколько записей DTLS, кодируемых последовательно. Поля Length из записей DTLS можно использовать для определения границ между записями. В последней записи это поле может отсутствовать. Первый байт содержимого дейтаграммы (payload) **должен** быть началом записи. Запись **недопустимо** разделять по разным дейтаграммам.

Записи DTLS без CID не включают идентификаторов ассоциаций и приложения должны обеспечивать мультиплексирование между ассоциациями. При работе по протоколу UDP можно использовать хост и номер порта при поиске подходящей ассоциации для входящих записей без CID.

Некоторые транспортные протоколы, такие как DCCP [RFC4340], используют свои порядковые номера. При использовании такого транспорта будет присутствовать 2 порядковых номера (DTLS и транспорт). Хотя это несколько снижает эффективность, номера в DTLS и транспорте служат разным целям, поэтому для концептуальной простоты сохранены оба номера.

Некоторые транспортные протоколы поддерживают контроль перегрузок для передаваемого трафика. Если окно перегрузки достаточно мало, повторы передачи при согласовании DTLS могут быть задержаны, что может приводить к тайм-аутам и ненужным повторам передачи. При использовании DTLS с таким транспортом следует соблюдать осторожность, чтобы не выйти за рамки вероятного окна перегрузки. В [RFC5238] описано отображение DTLS на DCCP с учётом этой проблемы.

4.4. Проблемы PMTU

В общем случае DTLS оставляет определение PMTU за приложениями, однако DTLS не может полностью игнорировать PMTU по трём причинам:

- «кадрирование» записей DTLS увеличивает размер дейтаграмм, снижая эффективное значение PMTU для приложений;
- в некоторых реализациях приложения не могут напрямую обращаться к сети, поэтому стек DTLS может получать сообщения ICMP Datagram Too Big [RFC1191] или ICMPv6 Packet Too Big [RFC4443];
- размер согласующих сообщений DTLS может превышать PMTU.

Для решения двух первых проблем уровню записей DTLS **следует** вести себя, как указано ниже. Если оценка PMTU доступна от базового транспорта, её следует сделать доступной для протоколов вышележащего уровня. В частности,

- для DTLS на основе UDP вышележащему протоколу **следует** разрешать получение оценки PMTU от уровня IP;
- для DTLS на основе DCCP вышележащему протоколу **следует** разрешать получение текущей оценки PMTU;
- для DTLS на основе TCP или SCTP, которые автоматически фрагментируют и собирают дейтаграммы, нет ограничений PMTU. Однако вышележащему уровню **недопустимо** делать записи размером более 2^{14} байтов.

Уровню записи DTLS **следует** также разрешать вышележащему протоколу определять величину расширения записи при обработке DTLS или он **может** сообщать вышележащему уровню значение PMTU от транспортного уровня за вычетом размера кадрирования записей DTLS.

Отметим, что в DTLS нет защиты от поддерльных сообщений ICMP и реализациям **следует** игнорировать такие сообщения, которые указывают PMTU ниже минимума для IPv4 и IPv6 - 576 и 1280 байтов, соответственно.

При наличии индикации транспортного уровня о превышении PMTU (через ICMP или отказ от передачи дейтаграммы, как указано в разделе 14 [RFC4340]) уровень записи DTLS **должен** информировать вышележащий уровень об ошибке.

Уровню записи DTLS **не следует** мешать вышележащим протоколам определять PMTU через [RFC1191] и [RFC4821] для IPv4 или через [RFC8201] для IPv6. В частности,

- если позволяет базовый транспорт, вышележащему протоколу **следует** разрешать устанавливать бит запрета фрагментирования (Don't Fragment или DF) для IPv4 или запрещать локальную фрагментацию для IPv6;
- если базовый транспорт (например, DCCP) разрешает приложению запрашивать зондирование PMTU, уровню записи DTLS **следует** удовлетворять такие запросы.

Ещё один вопрос чвязан с протоколом согласования DTLS. С точки зрения уровня записи DTLS это просто протокол вышележащего уровня, однако согласование DTLS происходит нечасто и занимает лишь несколько интервалов кругового обхода, поэтому обработка PMTU протокола согласования отдаёт предпочтение быстрому завершению процедуры, а не точному определению PMTU. Чтобы соединения создавались при таких условиях, реализациям DTLS **следует** соблюдать приведённые ниже правила.

- Если уровень записи DTLS сообщает уровню согласования DTLS, что сообщение слишком велико, уровню согласования **следует** сразу же попытаться фрагментировать сообщением с использованием имеющихся сведений о PMTU.
- Если повтор передачи на приводит к отклику, а значение PMTU неизвестно, в дальнейших повторах передачи **следует** снижать размер записи, соответствующим образом фрагментируя сообщение согласования. Эта спецификация не задаёт точного числа повторов передачи перед сокращением размера, но значение 2 или 3 будет подходящим.

4.5. Защита содержимого записи

Подобно TLS, протокол DTLS передаёт данные в форме последовательности защищённых записей, как описано ниже.

4.5.1. Предотвращение повторного использования

В каждой записи DTLS имеется порядковый номер для защиты от повторного использования (replay). Проверку порядковых номеров **следует** проводить с использованием описанной ниже процедуры скользящего окна, заимствованной из параграфа 3.4.3 в [RFC4303]. Поскольку в каждой эпохе нумерация записей начинается заново, для каждой эпохи требуется своё скользящее окно.

Для счётчика принятых записей в начале использования каждой эпохи **должно** устанавливаться значение 0. Для каждой принятой записи получатель **должен** убедиться, что она содержит порядковый номер, отличающихся от полученных ранее в этой эпохе в течение срока действия ассоциации. Проверку **следует** выполнять после снятия

защиты записи, поскольку в ином случае отбрасывание записи может служить каналом синхронизации номера записи. Отметим, что расчёт полного номера записи по частичному остаётся возможным каналом синхронизации номера, но менее мощным по сравнению с каналом при снятии защиты.

Дубликаты отвергаются с помощью скользящего окна приёма (реализация окна определяется локально, но она должна обеспечивать описанную ниже функциональность). Получателю **следует** выбирать достаточно большое окно, чтобы можно было обработать правдоподобное нарушение порядка, которое зависит от скорости передачи данных. Получатель не сообщает размер окна отправителю.

«Правый» край окна представляет наибольший проверенный номер, полученный в данной эпохе. Записи с номерами меньше «левого» края окна отвергаются. Записи с номерами, попадающими в окно, проверяются ещё раз по списку полученных записей из этого окна. Эффективным способом этой проверки служит использование битовой маски, как описано в параграфе 3.4.3 [RFC4303]. Если запись попадает в окно и является новой (не дубликат) или находится правей окна, она считается новой.

Окно **недопустимо** обновлять по принятой записи, пока с неё не будет снята защита.

4.5.2. Обработка недействительных записей

В отличие от TLS, протокол DTLS устойчив к непригодным записям (например, с некорректным форматом, размером, MAC и т. п.). В общем случае непригодные записи **следует** просто отбрасывать, сохраняя ассоциацию, однако ошибки **можно** фиксировать в системном журнале для диагностики. Реализации, решившие при этом выдавать предупреждения, **должны** генерировать критические предупреждения, чтобы избежать атак, где злоумышленник неоднократно проверяет реализацию, чтобы понять её реагирование на разные типы ошибок. Отметим, что при работе DTLS по протоколу UDP поступающая так реализация будет очень чувствительна к DoS-атакам, поскольку подделка UDP очень проста. Поэтому генерация критических предупреждений **не рекомендуется** для такого транспорта в качестве средства повышения надёжности сервиса DTLS и исключения риска атак с передачей поддельного трафика посторонним узлам.

Если DTLS работает на основе устойчивого к подделкам трафика (например, SCTP с SCTP-AUTH), безопасней передавать оповещения, поскольку атакующему будет очень сложно подделать дейтаграмму, которую транспортный уровень не отвергнет.

Отметим, что отклонение непригодных дейтаграмм на уровне ниже уровня конечного автомата согласования, это не влияет на ожидающие таймеры повтора передачи.

4.5.3. Ограничения AEAD

В параграфе 5.5 [TLS13] заданы ограничения для числа записей, которые можно защитить с использованием одного набора ключей. Эти ограничения связаны с алгоритмами AEAD и применимы для DTLS. Реализациям **не следует** выходить за пределы разрешённого числа защищаемых записей для согласованного AEAD и **следует** инициировать смену ключей до достижения предела.

В [TLS13] не задано ограничений для AEAD_AES_128_CCM, но анализ в Приложении B показывает, что можно использовать предел в 2^{23} пакетов для обеспечения такой же защиты конфиденциальности, какая задана в TLS.

Ограничения на использование, заданные в TLS 1.3, относятся к защите от атак на конфиденциальность и применяются к успешным применениям защиты AEAD. Защита целостности при аутентифицированном шифровании зависит от ограничения числа попыток подделки пакетов. TLS обеспечивает это путём закрытия соединений после отказа при проверки подлинности любой из записей. DTLS игнорирует невозможность аутентифицировать пакет, разрешая множественные попытки подделки.

Реализации **должны** учитывать число принятых пакетов, не прошедших аутентификацию, с каждым ключом. Если это число превышает порог, установленный для применяемого AEAD, реализации **следует** сразу же закрыть соединение. Реализациям **следует** инициировать обновление ключей с помощью update_requested до достижения этого предела. После запуска обновления ключей прежние ключи можно будет отбросить при достижении предела без закрытия соединения. Применение ограничений снижает вероятность успешной подделки пакета злоумышленником (см. [AEBounds] и [ROBUST]).

Для AEAD_AES_128_GCM, AEAD_AES_256_GCM, AEAD_CHACHA20_POLY1305 предельным числом пакетов с отказом при аутентификации является 2^{36}. Отметим, что анализ [AEBounds] даёт большее значение для AEAD_AES_128_GCM и AEAD_AES_256_GCM, но эта спецификация рекомендует более жёсткое ограничение. Для AEAD_AES_128_CCM установлен предел числа не прошедших аутентификацию пакетов $2^{23,5}$ (см. Приложение B).

Для AEAD_AES_128_CCM_8 AEAD, используемого в TLS_AES_128_CCM_8_SHA256, не задано ограничение для числа записей, не прошедших аутентификацию, что повышает вероятность подделки и создаёт для реализации риска DoS-атак (см. Приложение B.3). Поэтому TLS_AES_128_CCM_8_SHA256 **недопустимо** применять в DTLS без дополнительной защиты от подделок. Реализации **должны** задавать пределы использования AEAD_AES_128_CCM_8 с учётом применяемых дополнительных механизмов защиты от подделок.

Любой шифронабор TLS, заданный для применения с DTLS, **должен** устанавливать ограничения на использование связанной функции AEAD, с которой связана защита как конфиденциальности, так и целостности. Т. е. переделы должны устанавливать число пакетов, которые можно аутентифицировать, и разрешённое число пакетов, не прошедших проверку подлинности, по достижении которых потребуется обновление ключей. Предоставление ссылки на анализ, обосновывающий выбор значений, позволяет установить ограничения для разных вариантов применения.

5. Протокол согласования DTLS Handshake

DTLS 1.3 использует согласующие сообщения и потоки TLS 1.3 с указанными ниже изменениями.

1. Для обработки потерь, нарушения порядка и фрагментации нужно изменить заголовок согласования.
2. Добавлены таймеры повторной передачи для обработки потери пакетов.

3. Добавлен тип содержимого ACK для надёжной доставки согласующих сообщений.

Кроме того, DTLS использует расширение TLS 1.3 cookie для проверки маршрутизации по пути возврата в процессе организации соединения. Это важный механизм предотвращения DoS-атак для протоколов на основе UDP в отличие от протоколов на основе TCP, где такую проверку обеспечивает протокол TCP в процессе организации соединения.

Реализации DTLS не используют режим совместимости TLS 1.3 (compatibility mode), описанный в Приложении D.4 к [TLS13]. Серверам DTLS **недопустимо** возвращать (echo) значение legacy_session_id, полученное от клиента, а конечным точкам **недопустимо** передавать сообщения ChangeCipherSpec.

В остальном формат сообщений, потоки и логика DTLS не отличаются от применяемых в TLS 1.3.

5.1. Противодействие DoS-атакам

Протоколы защиты дейтаграмм очень уязвимы для разных DoS-атак, среди которых особо важны указанные ниже.

1. Злоумышленник может потреблять избыточные ресурсы сервера, передавая серию запросов согласования, вынуждающих сервер выделять состояния, а иногда и выполнять сложные криптографические операции.

2. Злоумышленник может использовать сервер в качестве усилителя атак, передавая сообщения инициирования соединений с ложными адресами отправителя, принадлежащими жертве. Сервер будет передавать отклики по адресу жертвы, создавая лавинную атаку на неё. В зависимости от выбранных параметров отклики могут быть достаточно большими, например, сообщениями Certificate.

Чтобы противостоять обоим типам атак DTLS заимствует метод cookie из Photuris [RFC2522] и IKE [RFC7296]. Когда клиент передаёт своё сообщение ClientHello серверу, тот **может** ответить сообщением HelloRetryRequest. Это сообщение и расширение cookie определены в TLS 1.3. HelloRetryRequest содержит не учитывающее состояние значение cookie (см. параграф 4.2.2 в [TLS13]). Клиент **должен** передать новое сообщение ClientHello, добавив cookie как расширение. Сервер проверяет cookie и при совпадении выполняет согласование. Этот механизм заставляет клиента или злоумышленника принять cookie, сто осложняет DoS-атаки с подменой адресов IP. Механизм не обеспечивает защиты от DoS-атак, организованных с действительных адресов IP.

Спецификация DTLS 1.3 меняет способ обмена cookie по сравнению с DTLS 1.2. В DTLS 1.3 используется сообщение HelloRetryRequest, передающее значение cookie клиенту через расширение. Принявший cookie клиент использует это же расширение для включения cookie в последующее сообщение ClientHello. В DTLS 1.2 применяется отдельное сообщение HelloVerifyRequest для передачи cookie клиенту без использования расширений. Для совместимости с прежними версиями поле cookie в ClientHello протокола DTLS 1.3 присутствует, но игнорируется поддерживающими DTLS 1.3 реализациями серверов.

Процесс обмена показан на рисунке 6 с указанием лишь cookie (без других расширений).

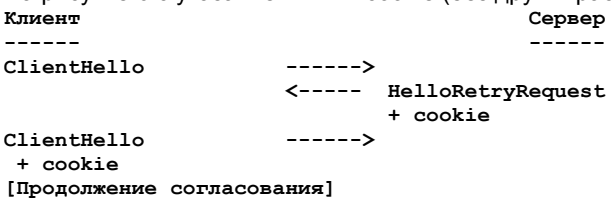

Рисунок 6. Обмен DTLS с сообщением HelloRetryRequest, содержащим cookie.

Расширение cookie определено в параграфе 4.2.2 [TLS13]. При передаче начального ClientHello у клиента ещё нет cookie. В этом случае расширение опускается и в поле legacy_cookie сообщение ClientHello **должен** указываться вектор размера 0 (1 байт в поле размера со значением 0). При ответе на HelloRetryRequest клиент **должен** создать новое сообщение ClientHello в соответствии с параграфом 4.1.2 в [TLS13].

При использовании HelloRetryRequest начальное сообщение ClientHello и HelloRetryRequest включаются в расчёт хэша «стенограммы» (transcript). Расчёт хэша сообщения HelloRetryRequest выполняется в соответствии с параграфом 4.4.1 в [TLS13]. Стенограмма согласования не сбрасывается при получении второго ClientHello и реализация серверного cookie без учёта состояния требует сохранения содержимого или хэш-значения исходного ClientHello (и HelloRetryRequest) в cookie. Исходное сообщение ClientHello включается в стенограмму согласования как синтетическое сообщение message_hash, поэтому для завершения согласования нужно лишь хэш-значение, хотя требуется полное содержимое HelloRetryRequest. При получении второго ClientHello сервер может проверить cookie и возможности получения клиентом пакетов по данному адресу IP. Если видимый IP-адрес встроен в cookie, это не позволит атакующему создать приемлемое сообщение ClientHello от другого пользователя.

При такой схеме возможна атака, где злоумышленник собирает много значений cookie с разных адресов, где он контролирует конечные точки, а затем может использовать их для атаки на сервер. Сервер может защититься от такой атаки, часто меняя значение секрета, что сделает собранные cookie непригодными. Если сервер хочет разрешить легитимным клиентам согласование через смену секрета (например, клиент получает cookie с Secret 1, затем передаёт второе сообщение ClientHello уже после перехода сервера на Secret 2), он может задать ограниченное окно, в котором воспринимаются оба секрета. В [RFC7296] предложено добавлять в таких случаях идентификатор ключа в cookie. Другим вариантом является просто проверка для обоих секретов. Серверам **рекомендуется** реализовать схему ротации ключей, которая позволит поддерживать ключи с перекрывающимися сроками действия. Как вариант, сервер может включать в cookie временную метку и отвергать cookie, созданные вне заданного интервала времени.

Серверам DTLS **следует** выполнять обмен cookie при каждом новом согласовании. Если сервер работает в среде, где атаки с усилением не создают проблем, например, используется ICE [RFC8445] для организации двухсторонней связности, сервер **может** отключить обмен cookie, однако этого не **следует** делать по умолчанию. Кроме того, сервер **может** отказаться от обмена cookie при возобновлении сессии, или, в более общем случае, при согласовании DTLS с применением обмена ключами на основе PSK и совпадении IP-адреса с одним из связанных с PSK. Серверы, обрабатывающие запросы 0-RTT и передающие отклики 0.5-RTT без обмена cookie, рискуют быть вовлечёнными в атаки с усилением, если размер исходящих сообщений существенно превышает размер получаемых. Серверу **следует** ограничивать объем данных, передаваемых по адресу клиента трёхкратным размером переданных клиентом данных,

пока сервер не будет уверен, что клиент способен принимать данные по этому адресу. Адрес клиента считается действительным после обмена cookie или завершения согласования. Клиенты **должны** быть готовы к обмену cookie при каждом согласовании. Отметим, что cookie действительны только в текущем согласовании и негодны для будущих.

Если сервер получает ClientHello с недействительным cookie, он **должен** прервать согласование с сигналом illegal_parameter. Это позволит клиенту перезапустить попытку соединения без cookie.

Как описано в параграфе 4.1.4 [TLS13], клиенты **должны** прерывать согласование с сигналом unexpected_message при получении второго сообщения HelloRetryRequest в том же соединении (т. е. когда сообщение ClientHello было ответом на HelloRetryRequest).

Клиентам DTLS, не желающим получать Connection ID, **следует** все равно предлагать расширение connection_id [RFC9146], пока профиль соединения не указывает иное. Это позволит серверу, желающему получать CID, согласовать это.

5.2. Формат сообщений DTLS Handshake

В DTLS применяются такие же сообщения Handshake как в TLS 1.3, однако перед отправкой их нужно преобразовать в DTLSHandshake с дополнительными данными для обработки потерь, изменения порядка и фрагментации.

```
enum {
    client_hello(1),
    server_hello(2),
    new_session_ticket(4),
    end_of_early_data(5),
    encrypted_extensions(8),
    request_connection_id(9),           /* Новое */
    new_connection_id(10),              /* Новое */
    certificate(11),
    certificate_request(13),
    certificate_verify(15),
    finished(20),
    key_update(24),
    message_hash(254),
    (255)
} HandshakeType;

struct {
    HandshakeType msg_type;    /* Тип согласования */
    uint24 length;             /* Число байтов в сообщении */
    uint16 message_seq;        /* Требуемое DTLS поле */
    uint24 fragment_offset;    /* Требуемое DTLS поле */
    uint24 fragment_length;    /* Требуемое DTLS поле */
    select (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;
        case request_connection_id: RequestConnectionId;
        case new_connection_id:     NewConnectionId;
    } body;
} DTLSHandshake;
```

В DTLS 1.3 стенограмма (transcript) сообщения рассчитывается для исходного сообщения TLS 1.3 Handshake без полей message_seq, fragment_offset, fragment_length. Это отличается от DTLS 1.2, где поля учитывались.

Первое сообщение, которая каждая из сторон передаёт в каждой ассоциации, имеет message_seq = 0, а при создании нового сообщения значение message_seq увеличивается на 1. При повторной передаче сообщения используется прежний номер без инкрементирования. С точки зрения уровня записи DTLS повторная передача является новой записью и эта запись будет иметь новое значение DTLSPlaintext.sequence_number.

> Примечание. В DTLS 1.2 значение message_seq сбрасывается в 0 при повторном согласовании. На первый взгляд повторное согласование в DTLS 1.2 похоже на обмен после согласования (post-handshake) в DTLS 1.3, однако в DTLS 1.3 message_seq не сбрасывается, что позволяет отличить повтор ранее переданного post-handshake от переданного недавно.

Реализации DTLS поддерживают (хотя бы номинально) счётчик next_receive_seq, для которого изначально устанавливается значение 0. Если При получении согласующего сообщения message_seq в нем совпадает с next_receive_seq, значение next_receive_seq увеличивается и сообщение обрабатывается. Если номер меньше next_receive_seq, сообщение **должно** отбрасываться, если же номер больше next_receive_seq, реализации **следует** поместить сообщение в очередь, но можно и отбросить (компромисс между ресурсами и пропускной способностью).

Кроме согласующих сообщений, отменённых спецификацией TLS 1.3, DTLS 1.3 отменяет HelloVerifyRequest, заданное исходно в DTLS 1.0. Совместимым с DTLS 1.3 реализациям **недопустимо** применять HelloVerifyRequest для проверки обратного маршрута. Клиенты с поддержкой DTLS 1.2 и DTLS 1.3 должны быть готовы взаимодействовать с серверами DTLS 1.2.

5.3. Сообщение ClientHello

Формат сообщений ClientHello в DTLS 1.3 отличается от применяемого в TLS 1.3 и показан ниже.

```
uint16 ProtocolVersion;
opaque Random[32];

uint8 CipherSuite[2];    /* Селектор шифронабора */

struct {
    ProtocolVersion legacy_version = { 254,253 }; // DTLSv1.2
    Random random;
    opaque legacy_session_id<0..32>;
    opaque legacy_cookie<0..2^8-1>;                // DTLS
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;
```

legacy_version

В прежних версиях DTLS это поле служило для согласования версии и указывало наибольшую версию, поддерживаемую клиентом. Опыт показал, что многие серверы не реализуют корректно согласование версий, что вело к «неприемлемости версии», когда сервер отвергал в остальном верные сообщения ClientHello, где указан номер версии выше поддерживаемого сервером. В DTLS 1.3 клиент указывает свои предпочтения версии в расширении supported_versions (см. параграф 4.2.1 в [TLS13]), а в поле legacy_version **должно** быть указано значение {254, 253}, которое было номером версии в DTLS 1.2. Записи supported_versions для DTLS 1.0 и DTLS 1.2 имеют значения 0xfeff и 0xfefd (чтобы соответствовать версии в линии). Значение 0xfefc указывает DTLS 1.3.

random

Как в TLS 1.3, за исключением того, что индикаторы версий, описанные в параграфе 4.1.3 [TLS13] для согласования с TLS 1.2, TLS 1.1 и ниже, применяются к DTLS 1.2 и DTLS 1.0, соответственно.

legacy_session_id

В TLS и DTLS до версии 1.3 поддерживалась функция восстановления сессии (session resumption), которая была объединена с PSK (pre-shared key) в версии 1.3. Клиенту с кэшированным идентификатором сессии, установленным сервером до DTLS 1.3, **следует** установить в этом поле данное значение. В иных случаях он **должен** указать в поле вектор нулевого размера (1-байтовое поле со значением 0).

legacy_cookie

Клиент, поддерживающий только DTLS 1.3, **должен** указывать поле legacy_cookie с размером 0. Если сообщение DTLS 1.3 ClientHello получено с иным значением этого поля, сервер **должен** прервать согласование с сигналом illegal_parameter.

cipher_suites

Как в TLS 1.3. Можно применять только шифры с DTLS-OK=Y.

legacy_compression_methods

Как в TLS 1.3.

extensions

Как в TLS 1.3.

5.4. Сообщение ServerHello

Сообщение DTLS 1.3 ServerHello отличается от TLS 1.3 лишь полем legacy_version = 0xfefd, указывающим DTLS 1.2.

5.5. Фрагментация и сборка сообщений Handshake

Как указано в параграфе 4.3, в одной дейтаграмме может передаваться одно или несколько сообщений согласования. Однако согласующие сообщения могут превышать по размеру возможности уровня транспортировки дейтаграмм. DTLS предоставляет механизм фрагментирования согласующего сообщения на несколько записей, каждая из которых может передаваться в отдельной дейтаграмме, что позволяет избежать фрагментации IP.

При передаче согласующего сообщения отправитель делит его на N смежных блоков данных, перекрытие которых **недопустимо**. Затем отправитель создаёт N сообщений DTLSHandshake, указывая в каждом то же значение message_seq, что и в исходном DTLSHandshake. В каждом новом сообщении указываются значения fragment_offset (число байтов, содержащихся в предыдущих фрагментах) и fragment_length (размер данного фрагмента). Поля length в каждом сообщении совпадают с полем length в исходном сообщении. Нефрагментированное сообщение является вырожденным случаем с fragment_offset=0 и fragment_length=length. Затем каждый фрагмент согласующего сообщения, помещённый в запись, **должен** быть доставлен в одной дейтаграмме UDP.

Когда реализация DTLS получает фрагмент согласующего сообщения, соответствующий ожидаемому порядковому номеру согласующего сообщения, она **должна** обработать фрагмент, буферизуя его в ожидании остальных фрагментов или сразу обрабатывая упорядоченные части сообщения. «Стенограмма» (transcript) включает полные сообщения TLS Handshake (собранные из фрагментов при необходимости). Отметим, что поля message_seq, fragment_offset, fragment_length удаляются при создании структуры Handshake.

Реализация DTLS **должна** быть способна обрабатывать перекрывающиеся фрагменты. Это позволит отправителю повторно передавать согласующие сообщения с меньшим размером фрагментов при изменении оценки PMTU. Отправителю **недопустимо** менять байты согласующего сообщения при повторе передачи. Получатели **могут** проверять идентичность переданных байтов и при наличии расхождений **следует** прерывать согласование с сигналом illegal_parameter.

Отметим, что как в TLS, в одну запись DTLS можно поместить несколько согласующих сообщений, если для них имеется место и они относятся к одной отправке (flight). Таким образом, два согласующих сообщения DTLS могут передаваться в одной или раздельных записях.

5.6. Сообщение EndOfEarlyData

Согласование DTLS 1.3 имеет одно важное отличие от согласования TLS 1.3 - сообщение EndOfEarlyData отсутствует как в линии, так и в стенограмме согласования. Поскольку записи DTLS включают эпоху, EndOfEarlyData не требуется для определения завершения ранних данных, а возможность потерь в DTLS позволяет злоумышленникам тривиально организовать атаки с удалением, которые EndOfEarlyData предотвращает в TLS. Серверам **не следует** воспринимать

записи из эпохи 1 бесконечно, как только они смогут обрабатывать записи из эпохи 3. Хотя нарушение порядка пакетов IP может приводить к прибытию записей из эпохи 1 после записей эпохи 3, это не может длиться долго по сравнению с временем кругового обхода. Сервер может отбросить ключи эпохи 1 после прихода первых данных эпохи 3 или сохранять в течение короткого времени для обработки записей эпохи 1 (см. 6.1. Значения эпохи и смена ключей).

5.7. Отправки сообщений DTLS Handshake

Согласующие сообщения DTLS группируются в серию отправок (flight), каждая из которых начинается передачей согласующего сообщения одним из партнёров и заканчивается приёмом ожидаемого отклика от другого. В таблице 1 приведён полный список комбинаций сообщений, составляющих отправку.

Таблица 1. Комбинации сообщений Handshake, создающих отправку (flight).

Клиент	***Сервер***	***Сообщения Handshake***
x		ClientHello
	x	HelloRetryRequest
	x	ServerHello, EncryptedExtensions, CertificateRequest, Certificate, CertificateVerify, Finished
x[1]		Certificate, CertificateVerify, Finished
	x[1]	NewSessionTicket

Необязательные сообщения в таблице 1 не выделены.

Ниже дано несколько примеров обмена, иллюстрирующих отправку с применением нотации [TLS13].

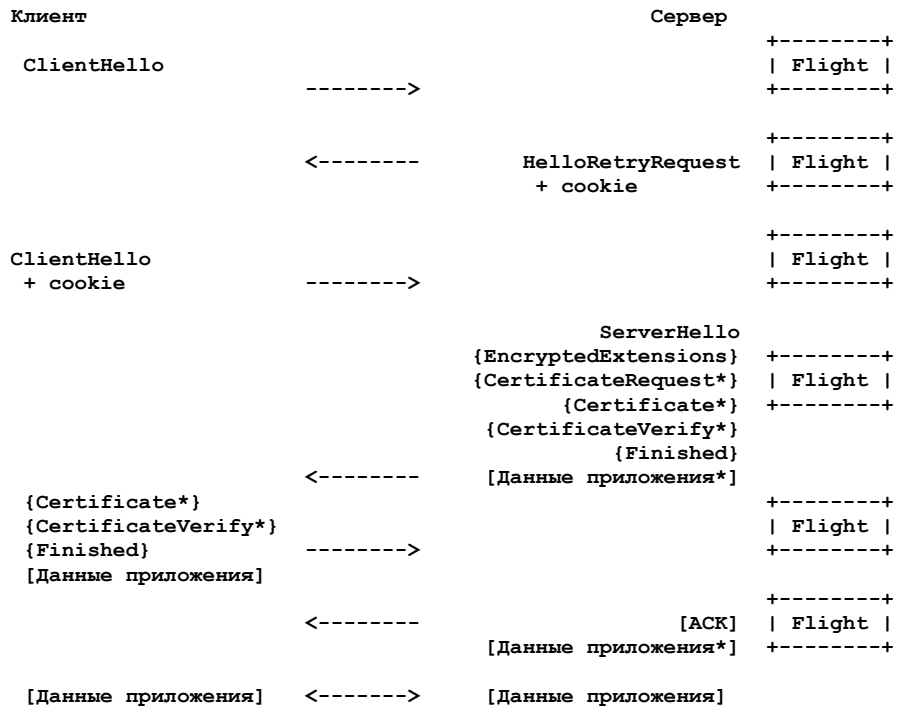

Рисунок 7. Отправка для полного согласования (с обменом Cookie).

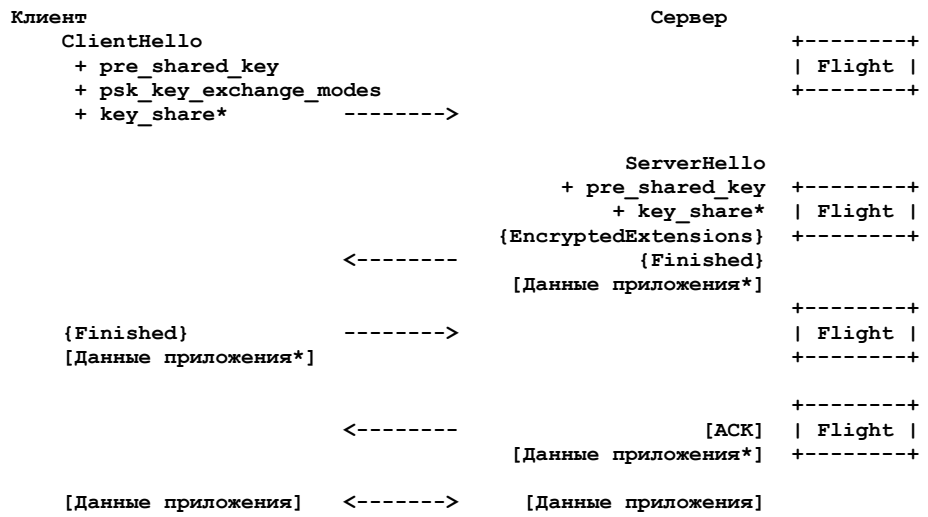

Рисунок 8. Отправка для возобновления и согласования PSK (без Cookie).

[1]Когда отправка при согласовании передаётся без ожидания ответа, как в случае с финальной отправкой клиента или сообщением NewSessionTicket, она должна подтверждаться сообщением ACK.

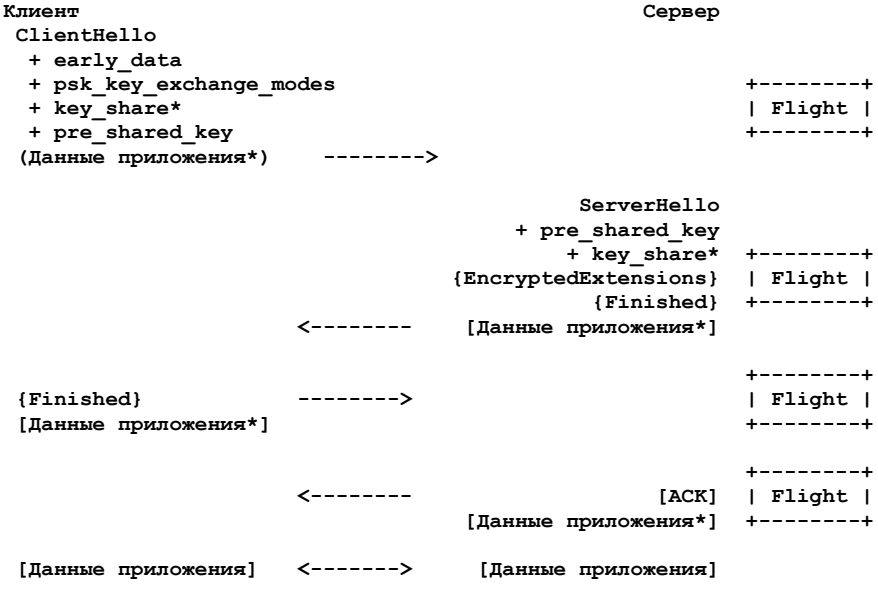

Рисунок 9. Отправка для согласования 0-RTT.

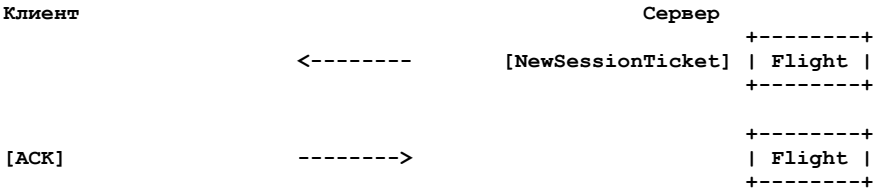

Рисунок 10. Отправка сообщения NewSessionTicket.

KeyUpdate, NewConnectionId, RequestConnectionId следуют схеме, похожей на NewSessionTicket, - одна сторона передаёт единственное сообщение, другая возвращает ACK.

5.8. Тайм-ауты и повтор передачи

5.8.1. Конечный автомат

DTLS использует простую схему с тайм-аутом и повтором передачи, конечный автомат которой показан на рисунке 11.

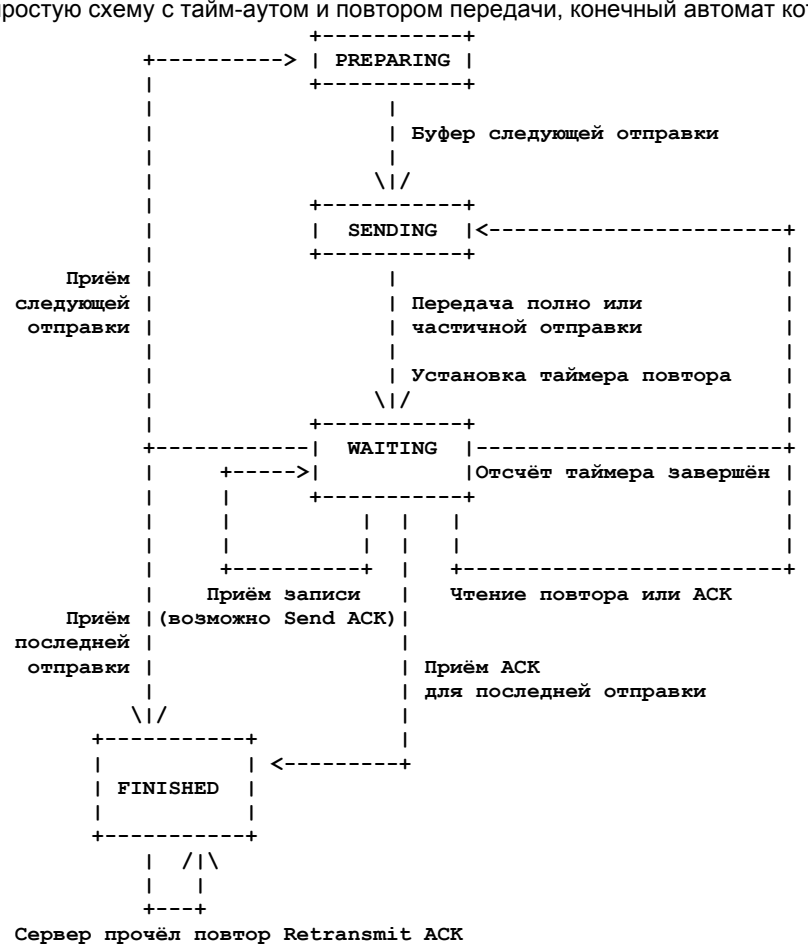

Рисунок 11. Конечный автомат тайм-аутов и повтора передачи DTLS.

Конечный автомат имеет 4 базовых состояния PREPARING, SENDING, WAITING, FINISHED.

В состоянии PREPARING реализация выполняет все расчёты, требуемые для следующей отправки сообщений, буферизует их для передачи (предварительно очищая буфер) и переходит в состояние SENDING.

В состоянии SENDING реализация передаёт буферизованную отправку (flight). При получении от партнёра одного или нескольких ACK (7. Сообщение ACK) реализации **следует** пропускать сообщения или фрагменты, которые были подтверждены. Когда сообщение передано, реализация запускает таймер повтора и переходит в состояние WAITING.

Из состояния WAITING возможны 4 варианта выхода, описанных ниже.

1. Завершение отсчёта таймера повтора - реализация переходит в состояние SENDING, повторяет отправку, корректирует и запускает таймер повтора (5.8.2. Значения таймеров) и возвращается в состояние WAITING.
2. Получение ACK от партнёра - при ACK для частичной отправки (7.1. Передача ACK) реализация переходит в состояние SENDING, повторяет неподтвержденную часть отправки, корректирует и снова запускает таймер повтора и возвращается в состояние WAITING. Если ACK подтверждает всю отправку, реализация отбрасывает в повторы и возвращается в состояние WAITING или, если ACK подтверждает финальную отправку, - в состояние FINISHED.
3. Реализация считывает повторно переданную отправку от партнёра, когда ни одно из переданных в ответ на эту отправку сообщений не было подтверждено - реализация преходит в состояние SENDING, повторяет отправку, корректирует и снова запускает таймер повтора и возвращается в состояние WAITING. Причина этого заключается в том, что получение дубликата сообщения вероятно является результатом завершения отсчёта таймера у партнёра, поэтому предполагается, что часть предыдущей отправки была потеряна.
4. Реализация получает часть или все сообщения следующей отправки - если это финальная отправка сообщений, реализация переходит в состояние FINISHED. Если реализации нужно передать новую отправку, она переходит в состояние PREPARING. Частично считывание (частичные сообщения или часть сообщений отправки) могут вызывать передачу реализацией сообщения ACK, как описано в параграфе 7.1.

Поскольку клиент DTLS передаёт первое сообщение (ClientHello), он начинает с состояния PREPARING. Серверы DTLS начинают с состояния WAITING, но с пустыми буферами и без таймера повтора.

Сервер в состоянии FINISHED **должен** отвечать на повторы передачи финальной отправки от клиента повтором подтверждения ACK в течение по меньшей мере удвоенного интервала MSL[1] [RFC0793].

Отметим, что из-за потери пакетов одна сторона может передавать данные приложения, даже если другая не получила от неё сообщение Finished. Реализации **должны** отбрасывать или буферизовать все записи с данными приложения для эпохи 3 и выше, пока не получат от партнёра сообщение Finished. Реализации **могут** считать получение данных сообщения с новой эпохой до приёма соответствующего сообщения Finished как свидетельство нарушения порядка или потери пакетов и сразу же передавать повторно свою финальную отправку, сокращая таймер повтора передачи.

5.8.2. Значения таймеров

Настройка таймера зависит от реализации и в некоторый средах требуется подстраивать таймер. Ошибки в обработке таймера могут вызывать серьёзные проблемы с перегрузкой, например, если у множества экземпляров DTLS тайм-аут возникает рано и они слишком быстро повторяют передачу в загруженный канал.

Если у реализации нет специфичной для развёртывания или внешних условий информации о времени кругового обхода, ей **следует** использовать для таймера исходное значение 1000 мсек (1 секунда) и удваивать его при каждом повторе передачи, вплоть до значения по меньшей мере 60 секунд (максимум, заданный в RFC 6298 [RFC6298]). Профили приложений **могут** рекомендовать другие значения, например, как указано ниже.

- Профили для конкретных сред развёртывания, таких как mesh-структуры со множеством пересылок (multi-hop) и слабым электропитанием, применяемых для IoT[2], **можно** задавать более долгий тайм-аут. Более подробное описание профиля DTLS 1.3 IoT приведено в [IOT-PROFILE].
- Протоколы, работающие в реальном масштабе времени, **могут** задавать более короткий тайм-аут. Для DTLS-SRTP [RFC5764] **рекомендуется** устанавливать по умолчанию 400 мсек, поскольку качество обслуживания клиентов снижается при односторонней задержке более 200 мсек и в таких системах столь длинные задержки маловероятны.

При наличии внешних сведений от RTT (например, от согласования ICE [RFC8445] или прежних соединений с тем же сервером) реализации **следует** устанавливать для таймера повтора передачи значение 1,5*RTT.

Реализациям **следует** сохранять текущее значение таймера, пока сообщение не будет передано и подтверждено без повтора передачи, после чего **следует** установить значение в 1,5 измеренных интервала кругового обхода для этого сообщения. После длительного бездействия (не менее 10 текущих значений таймера) реализация **может** сбросить таймер в исходное значение.

Поскольку повтор передачи предназначен для согласования, а не для потока данных, влияние коротких тайм-аутов на перегрузку меньше, чем в протоколах общего назначения, вроде TCP или QUIC. Опыт применения DTLS 1.2, где применяется более простой подход с повтором всего по тайм-ауту (retransmit everything on timeout) на практике не вызывает серьёзных проблем с перегрузкой.

5.8.3. Отправка большого размера

В DTLS нет встроенных механизмов контроля перегрузок и скорости. Обычно это не является проблемой, поскольку сообщения, как правило, невелики. Однако некоторые сообщения, в частности, Certificate, могут быть достаточно большими. Если все сообщения в одной большой отправке передаются разом, это может перегрузить сеть. Лучше передать лишь часть отправки с передачей дополнительных сообщений после подтверждения. Было стандартизовано несколько расширений для снижения размера сообщений Certificate, например, расширение cached_info [RFC7924] и сжатие сертификата [RFC8879] и [RFC6066], определяющее расширение client_certificate_url, позволяющее клиентам

[1]Maximum Segment Lifetime - максимальное время жизни сегмента TCP в сети. Произвольно выбрано значение 2 минуты.
[2]Internet of Things - Internet вещей.

DTLS передавать цепочку унифицированных указателей ресурсов (Uniform Resource Locator или URL) вместо своего сертификата.

Стеку DTLS **не следует** передавать более 10 в один приём.

5.8.4. Дублирование конечного автомата для сообщений после согласования

DTLS 1.3 использует указанные ниже категории сообщений после согласования (post-handshake).

1. NewSessionTicket
2. KeyUpdate
3. NewConnectionId
4. RequestConnectionId
5. Post-handshake client authentication

Сообщения каждой категории могут передаваться независимо, а надёжность обеспечивается независимыми конечными автоматами, поведение которых соответствует параграфу 5.8.1. Например, если сервер передаёт сообщения NewSessionTicket и CertificateRequest, создаётся два независимых конечных автомата.

Отправка нескольких экземпляров сообщений данной категории без завершений предшествующей транзакции разрешена лишь для некоторых категорий. В частности, сервер **может** передать сразу несколько сообщений NewSessionTicket, не ожидая ACK для переданных NewSessionTicket. Сервер **может** передать несколько сообщений CertificateRequest разом, не ожидая завершения предшествующих запросов аутентификации клиента. Однако **недопустимо** передавать сообщение KeyUpdate, NewConnectionId или RequestConnectionId, пока не подтверждено предыдущее сообщение того же типа.

> Примечание. За исключением аутентификации клиента после согласования, которое включает согласующие сообщения в обоих направлениях, прочие сообщения post-handshake представляют собой одну отправку и соответствующие конечные автоматы у отправителя сводятся к ожиданию ACK и повтору исходного сообщения. В частности, отметим, что сообщение RequestConnectionId не требует от получателя передавать в ответ NewConnectionId и оба эти сообщения обрабатываются независимо.

Создание и корректное обновление нескольких конечных автоматов требует обратной связи от логики согласования на уровень конечного автомата, указывающей, к какому конечному автомату относится каждое сообщение. Например, если сервере передаёт несколько сообщений CertificateRequest и получает в ответ Certificate, соответствующий конечный автомат можно определить только после проверки поля certificate_request_context. Точно так же при передаче сервером нескольких CertificateRequest и получении NewConnectionId в ответ решение об обработке NewConnectionId независимым конечным автоматом можно принять лишь после проверки типа согласующего сообщения.

5.9. Префикс криптографической метки

В параграфе 7.1 [TLS13] указано, что HKDF-Expand-Label использует префикс метки "tls13 ". Для DTLS 1.3 **нужно** указывать префикс "dtls13". Это позволяет различать ключи DTLS 1.3 и TLS 1.3. Отметим, что здесь нет пробела в конце, чтобы полный размер метки сохранялся в одной итерации хэша (DTLS на 1 символ длиннее, чем TLS).

5.10. Сообщения Alert

Отметим, что предупреждающие сообщения (alert) не передаются повторно даже в контексте согласования. Однако реализации DTLS, которая передаёт предупреждение, **следует** генерировать новое предупреждающее сообщение, если вызвавшая проблему запись приходит снова (например, как повтор согласующего сообщения). Реализациям **следует** обнаруживать, когда партнёр постоянно передаёт недопустимые сообщения и прерывать локальное состояние соединения после такого обнаружения. Отметим, что предупреждения передаются без гарантий и реализациям **не следует** полагаться на приём предупреждений для индикации ошибок или разрыва соединения.

Любые данные, принятые с парой эпоха-порядковый номер после действительного предупреждения о закрытии, **должны** игнорироваться. Отметим, что это отличается от протокола TLS 1.3, который зависит от порядка приёма, а не от эпохи и порядкового номера.

5.11. Организация новых ассоциаций с имеющимися параметрами

Если в DTLS пара клиент-сервер настроена так, что повторные соединения организуются с тем же квартетом адресов и портов, клиент может «молча» разорвать соединение и организовать другое с теми же параметрами (например, после перезагрузки). Сервер увидит это как новое согласование с epoch=0. В тех случаях, когда сервер считает, что у него есть ассоциация с данным квартетом адресов и портов, но при этом получает ClientHello с epoch=0, ему **следует** обработать новое согласование, но **недопустимо** разрушать имеющуюся ассоциацию, пока клиент не продемонстрирует достижимость путём завершения обмена cookie или полного согласования, включая доставку проверяемого сообщения Finished. После приёма корректного сообщения Finished сервер **должен** отказаться от прежней ассоциации во избежание конфликта между двумя действительными ассоциациями с перекрывающимися эпохами. Требование достижимости предотвращает атаки извне пути и вслепую для разрушения ассоциаций путём отправки поддельных сообщений ClientHello.

> **Примечание**. Не всегда возможно понять, к какой ассоциации относится запись. Например, если клиент выполняет согласование, отказывается от соединения, а затем сразу же начинает новое согласование, может оказаться невозможным определить, для какого соединения предназначена данная защищённая запись. В таких случаях может потребоваться пробная расшифровка, хотя реализации могут применять CID для исключения неоднозначности.

6. Пример согласования с тайм-аутом и повтором

Ниже приведён пример согласования с потерей пакетов и повтором передач. Отметим, что клиент передаёт пустое сообщение ACK, поскольку он может подтвердить запись Record 2, переданную сервером, лишь после того, как обработает сообщения из Record 0 для создания ключей эпохи 2, требуемых для шифрования и расшифровки

сообщений из 2. В разделе 7. Сообщение ACK обоснованы детали такого взаимодействия. Для простоты на рисунке 12 не сбрасываются номера записей, поэтому Record 1 на деле является Epoch 2, Record 0 и т. п..

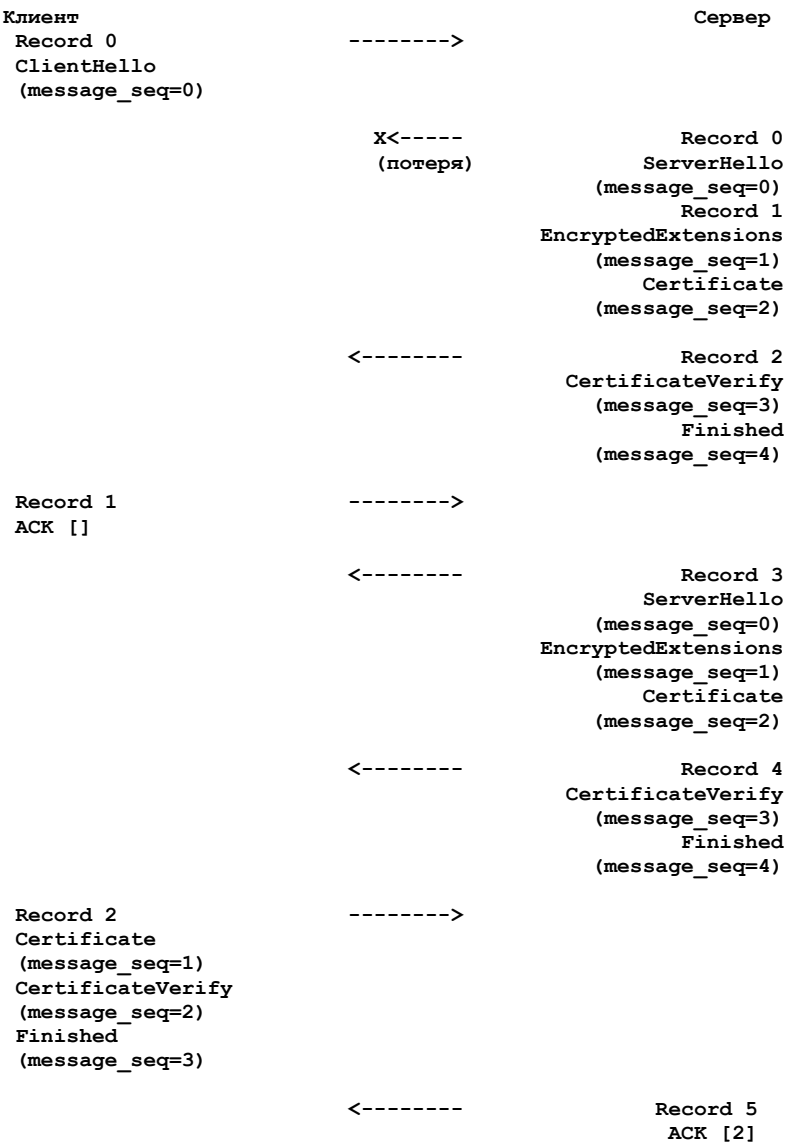

Рисунок 12. Пример обмена DTLS с потерей сообщения.

6.1. Значения эпохи и смена ключей

Получателю сообщения DTLS нужно выбрать корректный ключевой материал для обработки входящего сообщения. С учётом возможных потерь и нарушения порядка нужен идентификатор для определения шифронабора, применяемого для защиты содержимого записи. В DTLS для этого служит номер эпохи. В дополнение к заданным в TLS 1.3 этапам вывода ключей (раздел 7 в [TLS13]) получатель может пожелать сменить ключи в любой момент работы соединения. Поэтому он должен указать, что обновляет свои ключи, применяемые при передаче.

Эта версия DTLS назначает специальные номера эпохи для сообщений в протокольных обменах, позволяющие идентифицировать корректно состояние шифра.

- Эпоха 0 применяется с нешифрованными сообщениями DTLS - ClientHello, ServerHello, HelloRetryRequest.
- Эпоха 1 применяется с сообщениями, защищёнными с ключами, выведенными из client_early_traffic_secret. Отметим, что эта эпоха пропускается, если клиент не предоставляет ранних данных.
- Эпоха 2 применяется с сообщениями, защищёнными с помощью ключей, выведенных из [sender]_handshake_traffic_secret. Это сообщения, передаваемые в процессе начального согласования, такие как EncryptedExtensions, CertificateRequest, Certificate, CertificateVerify, Finished. Отметим, что сообщения после согласования защищаются с подходящим ключом трафика и не относятся к этой категории.
- Эпоха 3 применяется с сообщениями, защищёнными с ключами, выведенными из начального [sender]_application_traffic_secret_0. Это могут быть согласующие сообщения, такие как post-handshake (например, NewSessionTicket).
- Эпохи от 4 до 2^{64}-1 служат для данных, защищённых с использованием ключей, выведенных из [sender]_application_traffic_secret_N (N>0).

Используя эти номера эпох, получатель знает, какое состояние шифра использовалось для шифрования и защиты целостности сообщений. Реализациям, получающим записи с номером эпохи, для которого не определено соответствующее состояние шифра, **следует** обрабатывать их как записи, для которых не удалось снять защиту.

Отметим, что номера эпохи не переходят через максимум (wrap). Если реализация DTLS достигла максимального номера эпохи, она **должна** разорвать соединение. Расчёт ключей для трафика описан в параграфе 7.3 [TLS13]. На рисунке 13 показаны номера эпох для примера согласования DTLS.

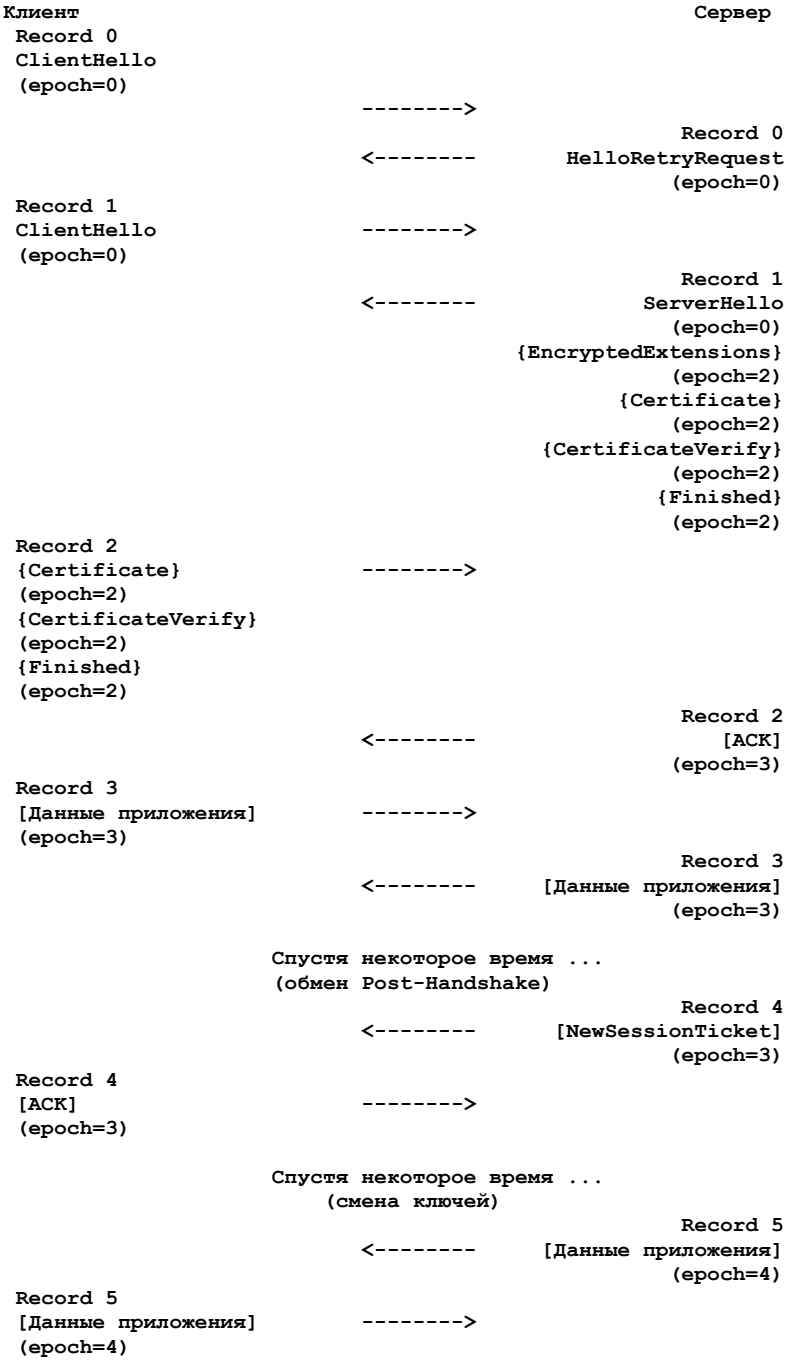

Рисунок 13. Обмен DTLS со сведениями об эпохе.

7. Сообщение ACK

Сообщения ACK применяются конечными точками для указания принятых и обработанных записей согласования от другой стороны. ACK не является согласующим сообщением, а представляет собой отдельный тип содержимого с кодом 26. Это позволяет не включать ACK в стенограмму согласования. Отметим, что ACK можно передавать в одной дейтаграмме UDP с записями согласования.

```
struct {
    RecordNumber record_numbers<0..2^16-1>;
} ACK;
```

record_numbers

Список записей, содержащих согласующие сообщения из текущей отправки, которые были приняты и обработаны или буферизованы по возрастанию номеров.

Реализациям **недопустимо** подтверждать записи, содержащие сообщения согласования или фрагменты, которые не были обработана или буферизованы, поскольку это может приводить в тупик. Например, реализациям **недопустимо** передавать ACK для сообщений согласования, отброшенных, не будучи следующими ожидаемыми сообщениями.

В процессе согласования ACK охватывают лишь текущую незавершённую отправку (это возможно, поскольку DTLS обычно выполняется по шагам). В частности, приём сообщения из согласующей отправки неявно подтверждает все предыдущие отправки. Соответственно, ACK от сервера не охватывает ClientHello и сообщение с клиентским сертификатом, поскольку ClientHello и Certificate от клиента относятся к разным отправкам. Реализации могут достигнуть этого, очищая свой список ACK при получении начала каждой отправки.

Для сообщений после согласования ACK **следует** передавать однократно для каждой принятой и обработанной записи согласования (возможно, задержанной) и они **могут** охватывать более одной отправки. Это включает записи с сообщениями, которые были отброшены по причине получения предыдущей копии.

В процессе согласования записи ACK **должны** передаваться с номером эпохи не меньше, мем номер в подтверждаемой записи. Требуется осторожность при обработке отправок, относящихся к разным эпохам. Например, если клиент получает только ServerHello и Certificate и хочет подтвердить (ACK) их в одной записи, он должен делать это с номером эпохи 2, поскольку требуется использовать номер эпохи не меньше 2 и пока нет возможности использовать больший номер. Реализациям **следует** просто использовать текущий наибольший номер переданной эпохи, который обычно является самым большим из доступных. После согласования реализации **должны** использовать наибольший доступный номер передающей эпохи.

7.1. Передача ACK

Когда реализация обнаруживает нарушение в приёме текущей входящей отправки, ей **следует** генерировать сообщение ACK, охватывающее сообщения из этой отправки, которые уже приняты и обработаны. Реализации могут по своему усмотрению определять признаки нарушений, но **рекомендуется** генерировать ACK в указанных случаях.

- Приём сообщения или фрагмента с нарушением порядка - не то сообщение, которое ожидалось следующим, или не та часть текущего сообщения.
- Получение части отправки без получения сразу же остальной части (которая может быть в той же дейтаграмме). Одним из подходов является установка для таймера 1/4 текущего значения таймера повтора при получении первой части и передача ACK по завершении отсчёта. Значение 1/4 выбрано достаточно произвольно. С учётом оценки времени кругового обхода при согласовании DTLS (обычно очень грубой или принятой по умолчанию) любое значение будет приблизительным и неизбежно нужен компромисс между повторной передачей изи слишком активных подтверждений (ACK) и чрезмерно активным тайм-аутом. Для сравнения, алгоритмы QUIC на основе алгоритма восстановления потерь (параграф 6.1.2 в [RFC9002]) работают с задержкой в 1/3 тайм-аута повторной передачи.

В общем случае отправки **должны** подтверждаться (ACK), если они не подтверждены неявно. В этой спецификации указанные ниже отправки неявно подтверждаются получением следующей отправки (обычно сразу же).

1. Отправки согласования, отличные от финальной отправки клиента в основном согласовании.
2. CertificateRequest от сервера после согласования.

Сообщения не следует передавать для таких отправок, где ответная отправка не может быть создана сразу же. Все остальные отправки **должны** подтверждаться ACK. В этом случае реализация **может** передавать явные ACK для полностью принятой отправки, даже если она в конечном итоге подтверждается неявно ответной отправкой. Ярким примером этого является аутентификация клиента в среде с ограничениями, где генерация сообщения CertificateVerify клиентом может занять много времени. Реализация **может** подтверждать записи, соответствующие каждой передаче каждой отправки или просто подтвердить последнюю. В общем случае реализациям **следует** подтверждать в ACK столько принятых пакетов, сколько можно поместить в запись ACK, так как это обеспечивает наиболее полные сведения и снижает вероятность ненужных повторов. Если пространство ограничено, реализации **следует** отдавать предпочтение записям, которые ещё не подтверждены.

> Примечание. Хотя некоторые сообщения post-handshake следуют шаблону запрос-отклик, это не обязательно предполагает получение. Например, сообщение KeyUpdate, переданное в ответ на KeyUpdate с request_update = update_requested, не подтверждает неявно предыдущее сообщение KeyUpdate, поскольку оба могли находится одновременно в сети.

ACK **недопустимо** передавать для записей с любым содержимым, кроме согласования, или записей, с которых невозможно снять защиту.

Отметим, что в некоторых случаях может потребоваться отправка ACK без номеров записей. Например, клиент может получить сообщение EncryptedExtensions до приёма ServerHello. Расшифровать EncryptedExtensions он не может, поэтому и не способен безопасно подтвердить сообщение (оно может быть повреждено). Если же клиент не передаст ACK, сервер в конечном итоге подтвердит первую отправку, но это может занять гораздо больше времени, чем фактический круговой обход между клиентом и сервером. Отправка клиентом пустого ACK сокращает этот процесс.

7.2. Получение ACK

При получении ACK реализации **следует** записать, что сообщения или фрагменты сообщений, переданные в записях, подтверждены этим ACK и исключить их из будущих повторов. При получении ACK, подтверждающего лишь часть сообщений из отправки, реализации **следует** повторить неподтвержденные сообщения или фрагменты. Это требует от реализации отслеживать включение сообщений в конкретные записи. После подтверждения всех сообщений из отправки реализация **должна** отметить все повторы передачи для этой отправки. Реализация **должна** считать запись подтверждённой, если та указана в любом ACK, это избавляет от ненужных повторов при очень больших отправках, когда получатель вынужден игнорировать подтверждения для записей, которые уже указаны в ACK. Как отмечено выше, приём любой записи, отвечающей за эту отправку, **должно** считаться неявным подтверждением всей отправки.

7.3. Обоснование решения

Сообщения ACK применяются в двух случаях:

- при признаках нарушения или отсутствии продвижения (progress);
- для индикации полного получения последней отправки в согласовании.

В первом случае применение ACK необязательно, поскольку партнёр повторит передачу в любом случае и ACK просто разрешает ускоренный или выборочный повтор, а не полный повтор всей отправки по тайм-ауту, как в прежних версиях DTLS. При использовании DTLS 1.3 в средах с потерями, таких как беспроводные сети со слабым питанием и протяжёнными радиоканалами, а также mesh-сети со слабым питанием рекомендуется применять ACK.

Во втором случае применение ACK обязательно для корректной работы протокола. Например, сообщение ACK, переданное клиентом на рисунке 13 подтверждает приём и обработку Record 4 (с сообщением NewSessionTicket) и при его отсутствии сервер будет повторять передачу NewSessionTicket, пока не будет достигнут предел числа повторов.

8. Обновление ключей

Как и в TLS 1.3, реализации DTLS 1.3 передают сообщение KeyUpdate, чтобы указать обновление своих ключей для передачи. Как и другие согласующие сообщения без «встроенного» отклика, KeyUpdate **должны** подтверждаться. Для упрощения реконструкции эпохи (4.2.2. Восстановление порядкового номера и эпохи) реализациям **недопустимо** передавать записи с новыми ключами или отправлять новое сообщение KeyUpdate, пока не подтверждено отправленное сообщение KeyUpdate (это позволит избежать излишних активных эпох).

Из-за потерь и нарушения порядка реализации DTLS 1.3 могут получать записи из прежней эпохи (приведённые выше требования исключают записи из более новой эпохи). Им **следует** пытаться обработать такие записи прежней эпохи (см. 4.2.2. Восстановление порядкового номера и эпохи), но **можно** просто отбрасывать их.

Из-за возможности потери ACK для сообщения KeyUpdate, не позволяющей отправителю KeyUpdate обновить свой ключевой материал, получатели **должны** сохранять прежний ключевой материал до получения и успешной расшифровки сообщения, использующего новые ключи.

На рисунке 14 показа обмен, иллюстрирующий обработку ACK для обновления ключей отправителем KeyUpdate, приводящего к смене эпохи.

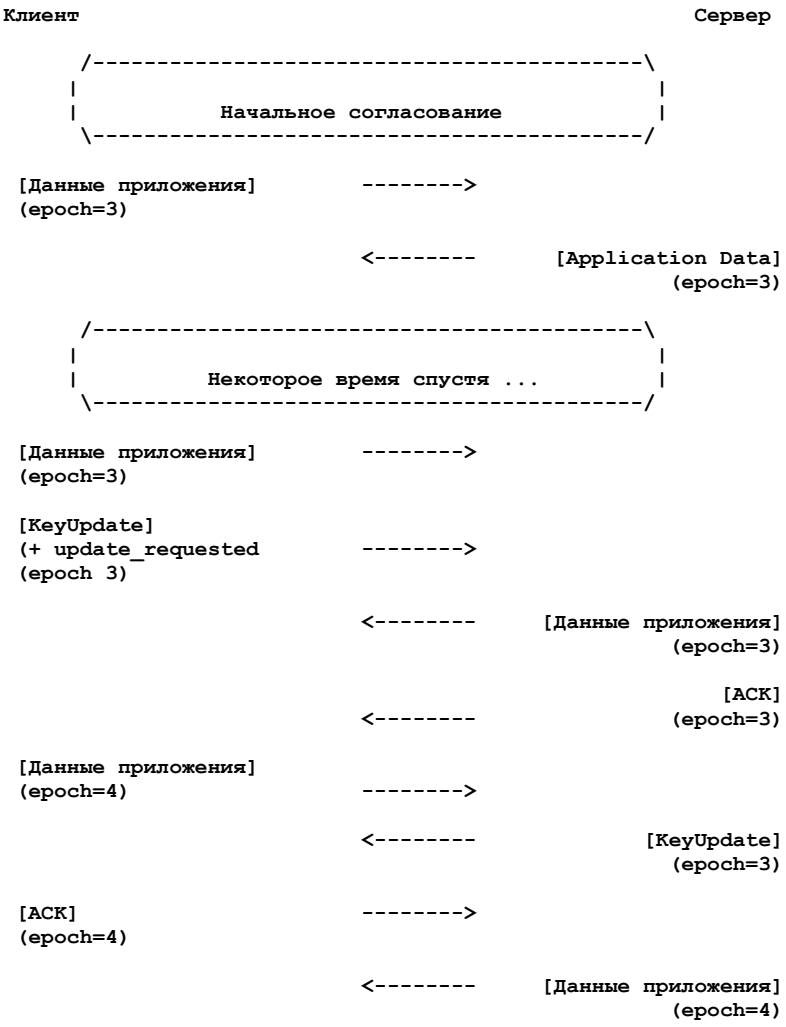

Рисунок 14. Пример обновления ключей DTLS.

При использовании 128-битового ключа, как в AES-128, смена ключей 2^{64} раз ведёт к высокой вероятности повтора ключей в данном соединении. Отметим, что даже при повторе ключей вектор инициализации (IV) создаётся независимо. Для обеспечения запаса безопасности передающей реализации **недопустимо** допускать превышение номером эпохи значения 2^{48}-1. Для продолжения смены эпох принимающим реализациям **недопустимо** форсировать применение этого правила. Если передающая реализация получает KeyUpdate с request_update = update_requested, ей **недопустимо** передавать своё сообщение KeyUpdate, если это приведёт к превышению указанного предела, и **следует** игнорировать флаг update_requested. Отметим, что это отличается от правила TLS 1.3 всегда передавать KeyUpdate в ответ на update_requested.

9. Обновление CID

Если клиент и сервер согласовали расширение connection_id [RFC9146], любой из них может передать желаемое значение CID другой стороне в сообщении NewConnectionId.

```
enum {
    cid_immediate(0), cid_spare(1), (255)
} ConnectionIdUsage;
```

```
opaque ConnectionId<0..2^8-1>;

struct {
    ConnectionId cids<0..2^16-1>;
    ConnectionIdUsage usage;
} NewConnectionId;
```

cids

Указывает набор CID, который отправитель предлагает использовать партнёру.

usage

Указывает, следует ли сразу же применять предложенные CID или они являются «запасными». Значение cid_immediate, указывает, что один из новых CID **должен** сразу же применяться для всех новых записей, значение cid_spare указывает, что **можно** использовать любой из имеющихся или новых CID.

Конечной точке **следует** использовать представленные получателем CID в порядке их указания. Реализации, получившие больше запасных CID, чем они хотят применять, **могут** просто отбросить лишние. Конечным точкам **недопустимо** иметь более одного остающегося сообщения NewConnectionId.

Реализациям, которые не согласовали расширение connection_id или согласовали получение пустого CID, **недопустимо** передавать NewConnectionId. Реализациям **недопустимо** передавать RequestConnectionId при отправке пустого Connection ID. Реализация, заметившая нарушение этих правил, **должна** прервать соединение с сигналом unexpected_message.

Реализациям **следует** применять новый CID при отправке по новому пути, а также **следует** запрашивать новые CID, если предполагается изменение пути.

```
struct {
  uint8 num_cids;
} RequestConnectionId;
```

num_cids

Желаемое число CID.

Конечным точкам **следует** отвечать на RequestConnectionId передачей NewConnectionId с cid_spare, содержащим число идентификаторов num_cids, как можно скорее. Конечным точка **недопустимо** передавать RequestConnectionId, когда имеющийся запрос ещё не выполнен, это предполагало бы запрос новых CID заблаговременно. Конечная точка **может** отвечать на запросы, которые она считает избыточными, сообщением NewConnectionId, содержащим число CID меньше num_cids, вплоть до отсутствия. Конечные точки **могут** отвечать на избыточные сообщения RequestConnectionId разрывом соединения с сигналом too_many_cids_requested (alert 52).

Конечным точкам **недопустимо** передавать такие соединения, если они не согласовали CID. Если реализация получает такое сообщение, когда CID не согласованы, она **должна** прервать соединение с сигналом unexpected_message.

9.1. Пример CID

Ниже приведён пример обмена для DTLS 1.3 с использованием 1 CID в каждом направлении.

Примечание. Расширение connection_id, применяемое в ClientHello и ServerHello, задано в [RFC9146].

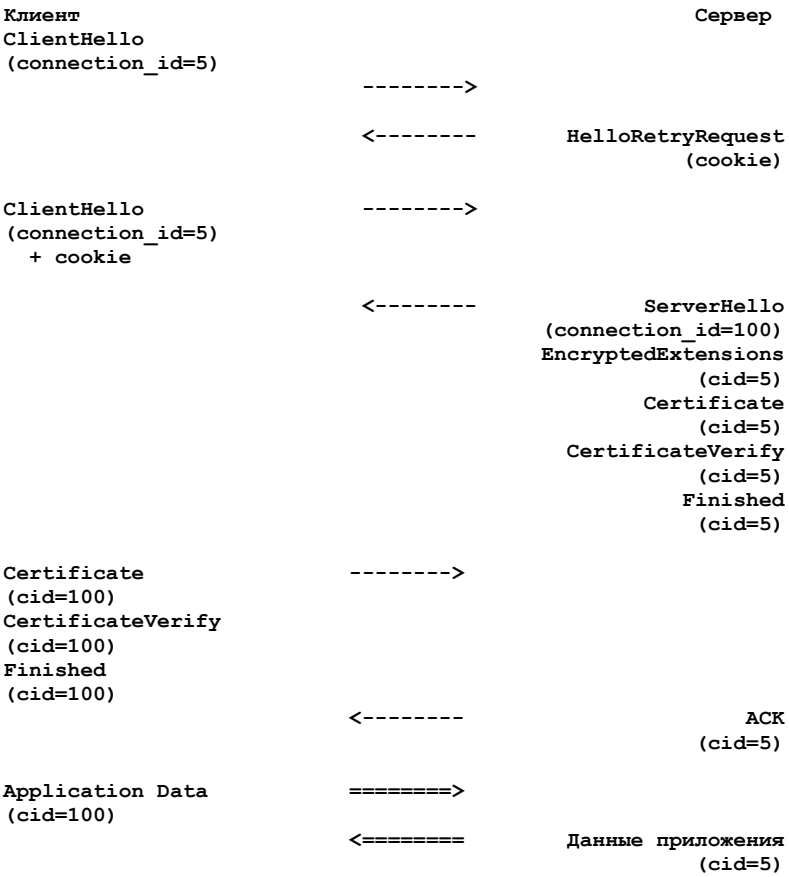

Рисунок 15. Пример обмена DTLS 1.3 с CID.

Если применение CID не согласовано, получатель **должен** отвергать любые записи, содержащие CID.

10. Протокол данных приложения

Сообщения с данными приложений передаются уровнем записей с разделением по записям и шифрованием в соответствии с текущим состоянием соединения. Сообщения обрабатываются как «прозрачные» для уровня записей данные.

11. Вопросы безопасности

Вопросы безопасности в основном рассмотрены в [TLS13].

Дополнительные соображения безопасности при использовании DTLS связаны в основном с DoS-атаками за счёт избыточного расхода ресурсов. DTLS включает обмен cookie, предназначенный для защиты от таких атак. Однако реализации, не применяющие этот механизм, остаются уязвимыми для DoS. В частности, серверы DTLS, не применяющие обмен cookie, могут использоваться как усилители атак, даже когда они сами не сталкиваются с отказами в обслуживании. Поэтому серверам DTLS **следует** использовать обмен cookie, если нет веских причин полагать, что атаки с усилением не актуальны в среде применения. Клиенты **должны** быть готовы к обмену cookie при каждом согласовании. Ниже перечислены некоторые важные свойства cookie, требуемые механизмом обмена, как описано в параграфе 3.3 [RFC2522]:

- Значение cookie **должно** зависеть от адреса клиента.
- **Недопустима** возможность создания cookie, считаемых действительными, кем-либо, кроме полномочного эмитента. Обычно это связано с проверкой целостности на основе секретного ключа.
- Генерацию и проверку cookie инициируют стороны, не прошедшие проверку подлинности, поэтому нужно ограничивать потребление ресурсов, чтобы механизм обмена cookie сам не стал частью DoS-атаки.

Хотя cookie должны разрешать серверу создавать корректную стенограмму (transcript) согласования, их **следует** создавать так, чтобы знания cookie не было досточно для воспроизведения содержимого ClientHello. Иначе могут возникнуть проблемы с будущими расширениями, такими как Encrypted Client Hello [TLS-ECH].

Хотя cookie создаются с использованием механизма аутентификации на основе ключа, следует предусматривать возможность смены соответствующего секретного ключа, чтобы временная компрометация ключа не создавала постоянного нарушения целостности механизма обмена cookie. Этот секрет не так ценен, как, например, ключ шифрования сеансовых квитанций (session-ticket-encryption), смена ключа создания cookie в таком же масштабе времени гарантирует, что ключи будут обновляться регулярно и обеспечат безопасную работу.

Обмен cookie обеспечивает проверку адресов при начальном согласовании. DTLS с CID позволяет менять адреса конечных точек в процессе работы ассоциации и такие адреса не охватываются обменом cookie при согласовании. Реализациям DTLS **недопустимо** менять адрес, на который они отправляют пакеты, при ответе на пакет с определённого адреса, пока они каким-либо способом не проверять доступность этого адреса. Данная спецификация не задаёт такой проверки, а в будущем потребуется полностью задать процедуру проверки. Но даже при такой проверке злоумышленник на пути передачи может отправить трафик в «черную дыру» или организовать атаку с отражением на стороннюю систему, поскольку у узлов DTLS нет возможности отличить подлинную смену адреса (например, в результате изменения привязки NAT) от злонамеренной подделки. Такие атаки вызывают беспокойство при значительной асимметрии размеров сообщений с запросами и откликами.

За исключением порядка и невоспроизводимости гарантии безопасности для DTLS 1.3 не отличаются от TLS 1.3. В TLS всегда обеспечивается упорядоченность и невоспроизводимость, но DTLS такой защиты не предоставляет.

В отличие от TLS, реализациям DTLS **не следует** отвечать на непригодные записи разрывом соединения.

TLS 1.3 требует защиты от воспроизведения для данных 0-RTT (точнее, для соединений с данными 0-RTT, см. раздел 8 в [TLS13]). DTLS предоставляет необязательный механизм защиты от воспроизведения на уровне отдельной записи, поскольку протоколы на основе дейтаграмм по своей природе подвержены нарушению порядка и повторному использованию пакетов (replay). Эти механизмы защиты от повторного использования ортогональны и ни один из них не соответствует требованиям другого.

Стенограмма согласования DTLS 1.3 не включает новые поля DTLS, поэтому имеет такой же формат, как в TLS 1.3. Однако стенограммы DTLS 1.3 и TLS 1.3 не пересекаются, поскольку используют разные номера версий. Кроме того, в планировании ключей DTLS 1.3 использует свою метку и будет создавать иные ключи для той же стенограммы.

Свойства безопасности и приватности CID для DTLS 1.3 основаны на описанных для DTLS 1.2 в [RFC9146], однако имеется несколько отличий, указанных ниже.

- В обеих версиях DTLS согласуется расширение для использования CID и их числа, а CID передаётся в заголовке записи DTLS (при согласовании). Однако способ включения CID в заголовок записи различается.
- Использование сообщения post-handshake позволяет клиенту и серверу обновлять свои CID, а конфиденциальность обмена защищается.
- Возможность использовать множество CID позволяет улучшить свойства приватности в многодомном случае. При использовании одного CID на разных путях от такого хоста злоумышленник может сопоставить взаимодействия по разным путям, что создаёт дополнительные проблемы приватности. Для решения этой проблемы реализациям **следует** пытаться использовать свежие CID при смене локального адреса или порта (хотя это не всегда можно обнаружить). Сообщение RequestConnectionId можно использовать для запроса у партнёра новых CID, чтобы иметь пул доступных CID.
- Механизм шифрования порядковых номеров (4.2.3. Шифрование номера записи) защищает от банального отслеживания злоумышленниками на пути, пытающимися сопоставить картины порядковых номеров на разных путях, что может происходить даже при использовании на этих путях разных CID, если порядковые номера не шифровать. Смена CID по событиям или регулярно помогает защититься от отслеживания злоумышленниками

на пути. Отметим, что шифрование номеров применяется для всех шифрованных записей DTLS 1.3 независимо от использования CID. Номера эпох не шифруются, поскольку они служат идентификаторами ключей, и это позволяет точнее сопоставлять пакеты одного соединения на разных путях через сеть.

- В DTLS 1.3 сообщения согласования шифруются раньше, чем в прежних версиях DTLS, поэтому злоумышленникам на пути доступно меньше сведений о клиенте DTLS (таких, как сертификат клиента).

12. Отличия от DTLS 1.2

Поскольку в TLS 1.3 внесено очень много изменений по сравнению с TLS 1.2, список различий между DTLS 1.2 и DTLS 1.3 также очень велик и ниже перечислены лишь наиболее важные из них.

- Новая модель согласования, сократившая обмен сообщениями.
- Поддерживаются только шифры AEAD, расчёт дополнительных данных упрощен.
- Исключена поддержка слабых и устаревших криптоалгоритмов.
- Сообщение HelloRetryRequest применяется в TLS 1.3 вместо HelloVerifyRequest.
- Более гибкое согласования шифронабора.
- Новый механизм возобновления сессий.
- Переопределена аутентификация PSK.
- Новая иерархия вывода ключей с новой конструкцией создания ключа.
- Улучшено согласование версий.
- Оптимизировано кодирование уровня записи и, следовательно, размеры.
- Добавлена функциональность CID.
- Порядковые номера шифруются.

13. Обновления, влияющие на DTLS 1.2

Этот документ вносит некоторые изменения, влияющие на реализации DTLS 1.2, даже не поддерживающие DTLS 1.3.

- Новый механизм защиты от понижения версии, описанный в параграфе 4.1.3 [TLS13] с применением в DTLS, как описано в параграфе 5.3. Сообщение ClientHello.
- Обновления, описанные в параграфе 1.3 [TLS13].
- Новые требования к соответствию, указанные в параграфе 9.3 [TLS13].

14. Взаимодействие с IANA

Агентство IANA выделило значение типа содержимого 26 из реестра TLS ContentType для сообщений ACK, заданных в разделе 7. В поле DTLS-OK указано значение Y. Типы содержимого 32-63 объявлены резервными и не выделяются.

Агентство IANA выделило значение 52 для предупреждения too_many_cids_requested из реестра TLS Alerts с указанием для DTLS-OK значения Y.

Агентство IANA выделило из реестра TLS HandshakeType, заданного в [TLS13], значения для идентификаторов request_connection_id (9) и new_connection_id (10), заданных этим документом с указанием для DTLS-OK значения Y.

Агентство IANA добавило ссылку на этот документ в реестр TLS Cipher Suites с приведённым ниже примечанием.

> Любой шифронабор TLS, заданный для использования с DTLS, **должен** указывать пределы использования связанной с ним функции AEAD, которая служит для защиты целостности и конфиденциальности, как указано в параграфе 4.5.3 RFC 9147.

15. Литература

15.1. Нормативные документы

[CHACHA] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9146] Rescorla, E., Ed., Tschofenig, H., Ed., Fossati, T., and A. Kraus, "Connection Identifier for DTLS 1.2", RFC 9146, DOI 10.17487/RFC9146, March 2022, <https://www.rfc-editor.org/info/rfc9146>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

15.2. Дополнительная литература

[AEAD-LIMITS] Günther, F., Thomson, M., and C. A. Wood, "Usage Limits on AEAD Algorithms", Work in Progress, Internet-Draft, draft-irtf-cfrg-aead-limits-04, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-aead-limits-04>.

[AEBounds] Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", 28 August 2017, <https://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf>.

[CCM-ANALYSIS] Jonsson, J., "On the Security of CTR + CBC-MAC", Selected Areas in Cryptography pp. 76-93, DOI 10.1007/3-540-36492-7_7, February 2003, <https://doi.org/10.1007/3-540-36492-7_7>.

[DEPRECATE] Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/info/rfc8996>.

[IOT-PROFILE] Tschofenig, H. and T. Fossati, "TLS/DTLS 1.3 Profiles for the Internet of Things", Work in Progress, Internet-Draft, draft-ietf-uta-tls13-iot-profile-04, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-uta-tls13-iot-profile-04>.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, DOI 10.17487/RFC2522, March 1999, <https://www.rfc-editor.org/info/rfc2522>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <https://www.rfc-editor.org/info/rfc4340>.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, DOI 10.17487/RFC4346, April 2006, <https://www.rfc-editor.org/info/rfc4346>.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, DOI 10.17487/RFC4347, April 2006, <https://www.rfc-editor.org/info/rfc4347>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, DOI 10.17487/RFC5238, May 2008, <https://www.rfc-editor.org/info/rfc5238>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <https://www.rfc-editor.org/info/rfc5763>.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <https://www.rfc-editor.org/info/rfc5764>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <https://www.rfc-editor.org/info/rfc7924>.

[RFC7983] Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", RFC 7983, DOI 10.17487/RFC7983, September 2016, <https://www.rfc-editor.org/info/rfc7983>.

[RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

[RFC8879] Ghedini, A. and V. Vasiliev, "TLS Certificate Compression", RFC 8879, DOI 10.17487/RFC8879, December 2020, <https://www.rfc-editor.org/info/rfc8879>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/info/rfc9002>.

[ROBUST] Fischlin, M., Günther, F., and C. Janson, "Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3", received 15 June 2020, last revised 22 February 2021, <https://eprint.iacr.org/2020/718>.

[TLS-ECH] Rescorla, E., Oku, K., Sullivan, N., and C.A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-14, 13 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14>.

Приложение A. Структуры данных и константы протокола

В этом приложении даны нормативные определения типов протоколов и констант.

A.1. Уровень Record

```
       struct {
           ContentType type;
           ProtocolVersion legacy_record_version;
           uint16 epoch = 0
           uint48 sequence_number;
           uint16 length;
           opaque fragment[DTLSPlaintext.length];
       } DTLSPlaintext;

       struct {
            opaque content[DTLSPlaintext.length];
            ContentType type;
            uint8 zeros[length_of_padding];
       } DTLSInnerPlaintext;

       struct {
           opaque unified_hdr[variable];
           opaque encrypted_record[length];
       } DTLSCiphertext;
```

```
0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|0|0|1|C|S|L|E E|
+-+-+-+-+-+-+-+-+
| Connection ID |
| (при наличии  |
/  согласуется  /   C   - присутствует Connection ID (CID)
|  размер)      |   S   - размер порядкового номера
+-+-+-+-+-+-+-+-+   L   - присутствует Length
| 8 или 16 битов|   E   - Epoch
|Sequence Number|
+-+-+-+-+-+-+-+-+
|16 битов Length|
| (при наличии) |
+-+-+-+-+-+-+-+-+
```

```
       struct {
           uint64 epoch;
           uint64 sequence_number;
       } RecordNumber;
```

A.2. Протокол Handshake

```
       enum {
           hello_request_RESERVED(0),
           client_hello(1),
           server_hello(2),
           hello_verify_request_RESERVED(3),
           new_session_ticket(4),
           end_of_early_data(5),
           hello_retry_request_RESERVED(6),
           encrypted_extensions(8),
           request_connection_id(9),           /* Новое */
           new_connection_id(10),              /* Новое */
           certificate(11),
           server_key_exchange_RESERVED(12),
           certificate_request(13),
           server_hello_done_RESERVED(14),
           certificate_verify(15),
           client_key_exchange_RESERVED(16),
           finished(20),
           certificate_url_RESERVED(21),
```

```
        certificate_status_RESERVED(22),
        supplemental_data_RESERVED(23),
        key_update(24),
        message_hash(254),
        (255)
    } HandshakeType;

    struct {
        HandshakeType msg_type;    /* Тип согласования */
        uint24 length;             /* Число байтов в сообщении */
        uint16 message_seq;        /* Требуемое DTLS поле */
        uint24 fragment_offset;    /* Требуемое DTLS поле */
        uint24 fragment_length;    /* Требуемое DTLS поле */
        select (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;
            case request_connection_id: RequestConnectionId;
            case new_connection_id:     NewConnectionId;
        } body;
    } Handshake;

    uint16 ProtocolVersion;
    opaque Random[32];

    uint8 CipherSuite[2];    /* Селектор шифронабора */

    struct {
        ProtocolVersion legacy_version = { 254,253 }; // DTLSv1.2
        Random random;
        opaque legacy_session_id<0..32>;
        opaque legacy_cookie<0..2^8-1>;               // DTLS
        CipherSuite cipher_suites<2..2^16-2>;
        opaque legacy_compression_methods<1..2^8-1>;
        Extension extensions<8..2^16-1>;
    } ClientHello;
```

A.3. ACK

```
    struct {
        RecordNumber record_numbers<0..2^16-1>;
    } ACK;
```

A.4. Управление идентификаторами соединений

```
    enum {
        cid_immediate(0), cid_spare(1), (255)
    } ConnectionIdUsage;

    opaque ConnectionId<0..2^8-1>;

    struct {
        ConnectionId cids<0..2^16-1>;
        ConnectionIdUsage usage;
    } NewConnectionId;

    struct {
      uint8 num_cids;
    } RequestConnectionId;
```

Приложение B. Анализ ограничений на использование CCM

В TLS [TLS13] и [AEBounds] не заданы пределы использования ключей для AEAD_AES_128_CCM. Однако для любого AEAD, применяемого с DTLS требуются ограничения на использование для защиты целостности и конфиденциальности. В этом приложении документируется анализ для алгоритма AEAD_AES_128_CCM.

В качестве основы для анализа служит [CCM-ANALYSIS]. Результаты анализа применяются для вывода ограничений применения, основанных на пределах, выбранных в [TLS13].

В анализе применяются символы умножения (*), деления (/) и возведения в степень (^), а также скобки для указания порядка действий. Специальные значения некоторых символов указаны ниже.

t

Размер тега аутентификации в битах. Для описываемого шифра t = 128.

n

Размер блочной функции в битах. Для описываемого шифра n = 128.

l

Число блоков в каждом пакете (см. ниже).

q

Число подлинных пакетов, созданных и защищённых конечными точками. Это значение ограничивает число пакетов, которые можно защитить без смены ключей.

v

Число поддельных пакетов, которые будет воспринимать конечная точка. Это значение ограничивает количество поддельных пакетов, которые конечная точка может воспринять без обновления ключей.

Анализ AEAD_AES_128_CCM основан на подсчете числа блочных операций, вовлечённых в создание каждого сообщения. Для простоты и соответствия анализу других функций AEAD в [AEBounds] предполагается размер пакета 2^10 блоков и предельный размер 2^14 байтов.

Для AEAD_AES_128_CCM общее число операций блочного шифра определяется суммой размера связанных данных в блоках, размера шифрованных данных в блоках и размера открытых данных в блоках плюс 1. В данном анализе это упрощено до удвоенного значения максимального размера записи в блоках (2l = 2^11). Это упрощение основано на том, что размер связанных данных ограничен одним блоком.

B.1. Ограничения для конфиденциальности

Для обеспечения конфиденциальности теорема 2 из [CCM-ANALYSIS] указывает, что злоумышленник получает заметное преимущество над идеальной псевдослучайной перестановкой (pseudorandom permutation или PRP) не более

```
(2l * q)^2 / 2^n
```

Для целевого преимущества 2^-60 в системе с 1 ключом, которое соответствует применяемому в TLS 1.3, как указано в [AEAD-LIMITS], это даёт соотношение

```
q <= 2^23
```

Таким образом, конечные точки не могут защитить более 2^23 с одним набором ключей, не дав атакующему преимущества выше целевого 2^-60.

B.2. Ограничения для целостности

Для обеспечения целостности теорема 1 из [CCM-ANALYSIS] указывает, что что злоумышленник получает заметное преимущество над идеальной PRP не более

```
v / 2^t + (2l * (v + q))^2 / 2^n
```

Целью является ограничение этого преимущества до значения 2^-57, соответствующего цели TLS 1.3, как указано в [AEAD-LIMITS]. Поскольку t и n имеют значение 128, первый член выражения пренебрежимо мал по сравнению со вторым и его можно удалить без существенного влияния на результат, что даёт в итоге

```
v + q <= 2^24,5
```

Используя ранее установленное значение 2^23 для q с округлением, получим верхний предел для v равный 2^23,5. Таким образом, конечные точки не могут аутентифицировать более 2^23,5 с одним набором ключей, не давая злоумышленнику преимущества выше целевого значения 2^-57.

B.3. Ограничения для AEAD_AES_128_CCM_8

Шифронабор TLS_AES_128_CCM_8_SHA256 использует функцию AEAD_AES_128_CCM_8 с коротким тегом аутентификации (t=64).

Пределы защиты конфиденциальности для AEAD_AES_128_CCM_8 совпадают с пределами AEAD_AES_128_CCM, поскольку они не зависят от размера тега (см. B.1. Ограничения для конфиденциальности).

Более короткий тег в 64 означает, что упрощения в B.2 не применимы для AEAD_AES_128_CCM_8. Если цель состоит в сохранении такого же запаса, как у других шифров, ограничения для подделки будут в значительной степени определяться первым членом выражения, т. е.

```
v <= 2^7
```

Поскольку это указывает попытки, не прошедшие проверку подлинности, применение такого ограничения может быть оправдано в некоторых средах. Однако его использование в реализациях общего назначения сделает соединения уязвимыми к недорогим DoS-атакам.

Этот анализ подтверждает мнение о непригодности TLS_AES_128_CCM_8_SHA256 для общего применения. В частности, TLS_AES_128_CCM_8_SHA256 не может применяться без дополнительных мер предотвращения подделки записей или ослабления влияния таких подделок. Это может потребовать понимания ограничений, имеющихся в конкретном развёртывании или приложении. Например, можно установить иное целевое значение для преимущества, получаемого злоумышленником, на основе понимания ограничений, вносимых конкретным применениемDTLS.

Приложение C. Подводные камни реализации

В дополнение к аспектам TLS, которые были источниками проблем совместимости и безопасности (Приложение C.3 к [TLS13]), DTLS вносит несколько своих источников возможных проблем, отмеченных ниже.

- Корректность обработки сообщений из разных эпох при смене ключей. Это включает определение нужного ключа и защиту от повторного использования, если она включена.
- Повторная передача согласующих сообщений не подтверждённых явно или неявно (5.8. Тайм-ауты и повтор передачи).
- Корректность обработки принятых фрагментов сообщений согласования, включая нарушение их порядка.
- Корректность обработки сообщений согласования, принятых с нарушением порядка, включая их буферизацию или отбрасывание.
- Ограничение объёма данных, передаваемых узлу до проверки его адреса.
- Проверка явного указания размера записи в дейтаграмме, где запись содержится.

Участники работы

Многие люди внесли вклад в предыдущие версии DTLS и были отмечены в прежних версиях спецификаций DTLS или упомянутых там документах.

Hanno Becker
Arm Limited
Email: Hanno.Becker@arm.com

David Benjamin
Google
Email: davidben@google.com

Thomas Fossati
Arm Limited
Email: thomas.fossati@arm.com

Tobias Gondrom
Huawei
Email: tobias.gondrom@gondrom.org

Felix Günther
ETH Zurich
Email: mail@felixguenther.info

Benjamin Kaduk
Akamai Technologies
Email: kaduk@mit.edu

Ilari Liusvaara
Independent
Email: ilariliusvaara@welho.com

Martin Thomson
Mozilla
Email: martin.thomson@gmail.com

Christopher A. Wood
Cloudflare
Email: caw@heapingbits.net

Yin Xinxing
Huawei
Email: yinxinxing@huawei.com

Концепция шифрования порядкового номера заимствована из QUIC [RFC9000]. Спасибо авторам RFC 9000 за их работу. Felix Günther и Martin Thomson внесли вклад в анализ, представленный в Приложении B. Спасибо Jonathan Hammell, Bernard Aboba, Andy Cunningham за их комментарии.

Кроме того, авторы благодарны за обхорные комментации членам IESG: Martin Duke, Erik Kline, Francesca Palombini, Lars Eggert, Zaheduzzaman Sarker, John Scudder, Éric Vyncke, Robert Wilton, Roman Danyliw, Benjamin Kaduk, Murray Kucherawy, Martin Vigoureux, Alvaro Retana.

Адреса авторов

Eric Rescorla
Mozilla
Email: ekr@rtfm.com

Hannes Tschofenig
Arm Limited
Email: hannes.tschofenig@arm.com

Nagendra Modadugu
Google, Inc.
Email: nagendra@cs.stanford.edu

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru