Internet Engineering Task Force (IETF)
Request for Comments: 9001
Category: Standards Track
ISSN: 2070-1721

M. Thomson, Ed.
Mozilla
S. Turner, Ed.
sn3rd

May 2021

Using TLS to Secure QUIC

Использование TLS для защиты QUIC

Аннотация

Этот документ описывает применение защиты на транспортном уровне (Transport Layer Security или TLS) для QUIC.

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.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9001.

Авторские права

Авторские права (Copyright (c) 2021) принадлежат IETF Trust и лицам, указанным в качестве авторов документа. Все права защищены.

Этот документ является субъектом прав и ограничений, перечисленных в BCP 78 и IETF Trust Legal Provisions и относящихся к документам IETF (http://trustee.ietf.org/license-info), на момент публикации данного документа. Прочтите упомянутые документы внимательно, поскольку в них описаны права и ограничения, относящиеся к данному документу. Фрагменты программного кода, включённые в этот документ, распространяются в соответствии с упрощённой лицензией BSD, как указано в параграфе 4.e документа Trust Legal Provisions, без каких-либо гарантий (как указано в Simplified BSD License).

Оглавление

[1]Internet Engineering Task Force.
[2]Internet Engineering Steering Group.

1. Введение

В этом документе описан способ защиты QUIC [QUIC-TRANSPORT] с использованием TLS [TLS13].

TLS 1.3 обеспечивает значительное снижение задержки при организации соединений по сравнению с предыдущими версиями. При отсутствии потери пакетов большинство новых соединений может быть организовано и защищено за один круговой обход, а в последующих соединениях между теми же клиентом и сервером клиент зачастую может передавать данные приложения сразу же, используя соединение без кругового обхода (zero round-trip setup).

Этот документ описывает использование TLS для защиты QUIC.

2. Соглашения о нотации

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.

В этом документе используются термины, определённые в [QUIC-TRANSPORT].

Для краткости аббревиатура TLS обозначает TLS 1.3, хотя могут применяться и более новые версии (параграф 4.2).

2.1. Обзор TLS

TLS обеспечивает паре конечных точек способ организации связи через среду без доверия (например, Internet). TLS обеспечивает проверку подлинности партнёров, а также защиту конфиденциальности и целостности сообщений, передаваемых между партнёрами.

TLS представляет собой многоуровневый протокол, структура которого показана на рисунке 1.

Каждое сообщение уровня содержимого (например, согласование - handshake, сигналы, данные приложения) передаётся в серии типизованных записей уровня записи TLS. Каждая запись защищена криптографически и передаётся через транспорт с гарантией доставки (обычно TCP) и сохранением порядка.

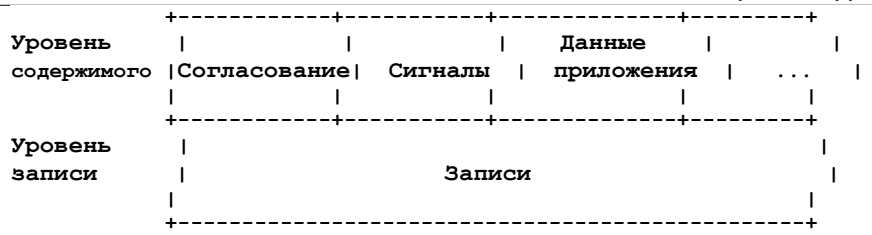

Рисунок 1. Уровни TLS.

Между конечными точками (клиент и сервер) выполняется аутентифицированный обмен ключами. Обмен инициирует клиент, а сервер отвечает ему. При успешном обмене ключами клиент и сервер согласуют секрет. TLS поддерживает заранее распространённые общие ключи (pre-shared key или PSK) и обмен Diffie-Hellman по конечным полям или эллиптическим кривым ((EC)DHE). PSK служит основой для начальных данных (Early Data, 0-RTT), а обмен - совершенную защиту (forward secrecy или FS) при уничтожении ключей (EC)DHE. Режимы можно комбинировать для обеспечения совершенной защиты при аутентификации PSK.

По завершении согласования TLS клиент узнает и подтвердит отождествление сервера, а сервер сможет узнать и отождествить клиента. TLS поддерживает аутентификацию на основе сертификатов X.509 [RFC5280] для клиента и сервера. При использовании обмена ключами PSK (как при возобновлении) знание PSK служит для аутентификации партнёра.

Обмен ключами TLS устойчив к вмешательству злоумышленников и создаёт общие секреты, которые не могут контролироваться ни одним из участвующих партнёров.

TLS обеспечивает два основных режима согласования для QUIC.

- Полное согласование 1-RTT, в котором клиент может передавать данные после одного кругового обхода, а сервер отвечает сразу после получения от клиента согласующего сообщения.
- Согласование 0-RTT, где клиент использует полученные ранее от сервера сведения для немедленной отправки данных приложения. Эти данные могут быть воспроизведены (replay) атакующим, поэтому 0-RTT не подходит для передачи инструкций, способных вызвать какие-либо действия с нежелательными последствиями.

Упрощённая схема согласования TLS с данными приложения 0-RTT показана на рисунке 2.

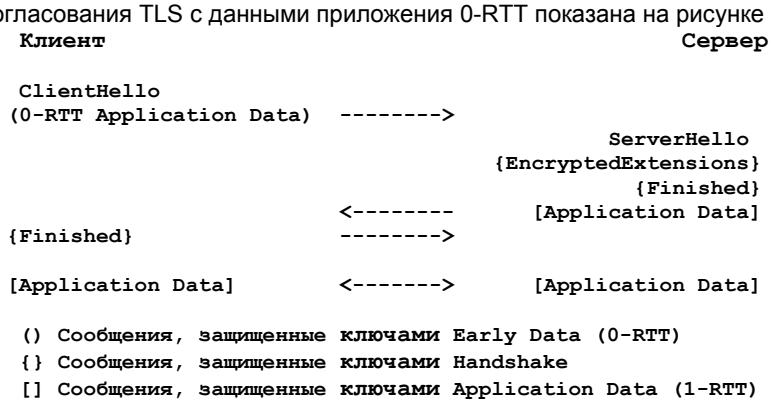

Рисунок 2. TLS Handshake с 0-RTT.

На рисунке 2 не показано сообщение EndOfEarlyData, которое не применяется в QUIC (см. параграф 8.3), как и сообщения ChangeCipherSpec и KeyUpdate. Сообщение ChangeCipherSpec избыточно в TLS 1.3 (параграф 8.4). В QUIC имеется свой механизм обновления ключей, описанный в разделе 6. Обновление ключей.

Данные защищаются с использованием нескольких уровней шифрования:

- ключи Initial;
- ключи 0-RTT;
- ключи Handshake;
- ключи 1-RTT.

Данные приложений могут присутствовать на уровнях 0-RTT и 1-RTT, сообщения согласования и сигналов - на любом.

Согласование 0-RTT может применяться, если между клиентом и сервером уже было взаимодействие. В согласовании 1-RTT клиент не может передавать защищённые данные приложения, пока сервер не передаст сообщения Handshake.

3. Обзор протокола

QUIC [QUIC-TRANSPORT] предполагает ответственность за защиту конфиденциальности и целостности пакетов. Для этого протокол использует ключи, выведенные из согласования TLS [TLS13], но вместо передачи записей TLS по протоколу QUIC (как в TCP) сообщения согласования и сигналы TLS передаются напрямую через транспорт QUIC, принимающий на себя обязанности уровня записи TLS, как показано на рисунке 3.

QUIC также полагается на TLS для проверки подлинности и согласования параметров, важных для конфиденциальности и производительности. Вместо строго деления по уровням эти два протокола взаимодействуют - QUIC использует согласование TLS, а TLS - гарантированную упорядоченную доставку и уровень записи, предоставляемые протоколом QUIC. На высоком уровне происходят два основных взаимодействия между TLS и QUIC:

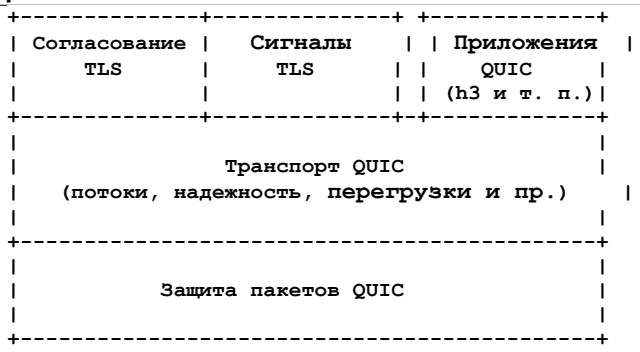

Рисунок 3. Уровни QUIC.

- TLS передаёт и принимает сообщения через QUIC с обеспечением абстракции надёжного потока для TLS;
- TLS обеспечивает серию обновлений для QUIC, включая (a) новые ключи защиты пакетов и (b) смену состояний, таких как завершения согласования, сертификат сервера и т. п.

На рисунке 4 более подробно показаны эти взаимодействия, где защита пакетов QUIC вызывается отдельно.

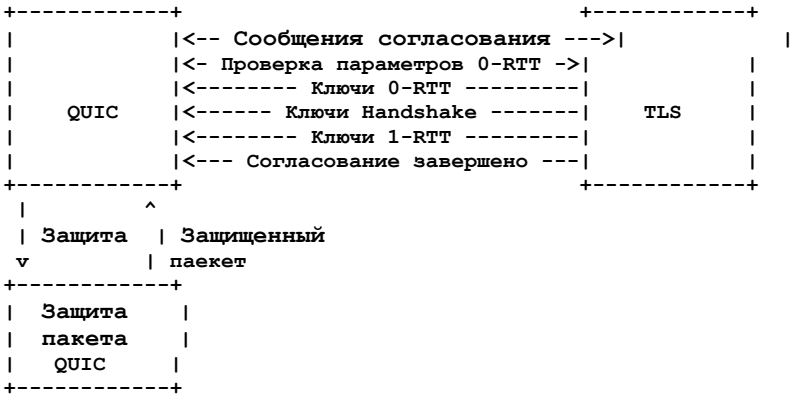

Рисунок 4. Взаимодействие QUIC и TLS.

В отличие от TLS по протоколу TCP, приложения QUIC, желающие передавать данные, не шлют их с использованием записей TLS Application Data, а просто передают кадры QUIC STREAM или иных типов, которые доставляются в пакетах QUIC.

4. Передача сообщений TLS

QUIC передаёт данные согласования TLS в кадрах CRYPTO, каждый из которых содержит непрерывный блок данных согласования, указанных смещением и размером. Эти кадры помещаются в пакеты QUIC и шифруются с текущим уровнем. Как и в TLS по протоколу TCP, после доставки данных согласования TLS протоколу QUIC, тот принимает на себя ответственность за их доставку. Каждый блок данных, созданный TLS, связывается с набором ключей, которые TLS применяет в данный момент. Если протоколу QUIC требуется повторно передать данные, он **должен** использовать те же ключи даже в случае перехода TLS на новые ключи.

Каждый уровень шифрования соответствует пространству номеров пакетов. Используемое пространство номеров определяет семантика кадра. Некоторые кадры запрещены в разных пространствах номеров (см. параграф 12.5 в [QUIC-TRANSPORT]).

Поскольку порядок пакетов может меняться при передаче, QUIC использует тип пакета для указания ключей, использованных для защиты данного пакета, как показано в таблице 1. Когда нужно передать пакеты разных типов, конечным точкам **следует** объединять их для отправки в одной дейтаграмме UDP.

Таблица 1. Ключи шифрования по типам пакетов.

Тип пакета	***Ключи шифрования***	***Пространство номеров***
Initial	Initial secrets	Initial
0-RTT Protected	0-RTT	Application data
Handshake	Handshake	Handshake
Retry	Retry	N/A
Version Negotiation	N/A	N/A
Short Header	1-RTT	Application data

В разделе 17 [QUIC-TRANSPORT] показано использование пакетов с разным уровнем шифрования при согласовании.

4.1. Интерфейс с TLS

Как показано на рисунке 4, интерфейс из QUIC в TLS поддерживает четыре основных функции:

- передача и приём сообщений в процессе согласования;
- обработка сохранённого состояния транспорта и приложения из возобновляемой сессии и определение их пригодности для генерации или восприятия данных 0-RTT;
- смена ключей (отправка и приём);
- обновление статуса согласования.

Для настройки TLS могут потребоваться дополнительные функции. В частности, QUIC и TLS нужно согласовать ответственность за проверку свидетельств партнёра, такую как проверка сертификатов [RFC5280].

4.1.1. Завершение согласования

В этом документе согласование TLS считается завершённым, когда об этом сообщает стек TLS. Это происходит, когда стек TLS передал сообщение Finished и проверил сообщение Finished от партнёра. Проверка партнерского сообщения Finished обеспечивает конечной точке гарантию неизменности предшествующих сообщения согласования. Отметим, что точки не завершают согласование одновременно, поэтому любые требования, основанные на завершении согласования, зависят от соответствующей конечной точки.

4.1.2. Подтверждение согласования

В этом документе согласование TLS считается подтверждённым на сервере после его завершения. Сервер **должен** передать кадр HANDSHAKE_DONE, как только согласование завершится. Клиент считает согласование завершённым после получения кадра HANDSHAKE_DONE.

Кроме того, клиент может считать согласование завершённым, получив подтверждение для пакета 1-RTT. Это может быть реализовано путём записи наименьшего порядкового номера пакета, переданного с ключами 1-RTT, и сравнения его с Largest Acknowledged в любом принятом кадре 1-RTT ACK - большее значение последнего подтверждает согласование.

4.1.3. Отправка и получение сообщений Handshake

При управлении согласованием TLS зависит способности передавать и принимать сообщения Handshake. На этом интерфейсе имеются две основных функции - запрос сообщений от QUIC и предоставление протоколом QUIC байтов, образующих сообщения Handshake.

Перед началом согласования QUIC предоставляет протоколу TLS транспортные параметры (см. параграф 8.2), которе он хочет передать. Клиент QUIC запускает TLS, запрашивая байты согласования TLS у протокола TLS, которые он получает до отправки своего первого пакета. Сервер QUIC запускает процесс, предоставляя протоколу TLS байты согласования от клиента.

Стек TLS на конечной точке в любой момент будет иметь текущий уровень шифрования передачи и приёма. Уровни шифрования TLS определяют тип пакета QUIC и применяемые для защиты данных ключи. Каждый уровень шифрования связан со своей последовательностью байтов, которая гарантированно передаётся партнёру в кадрах CRYPTO. Когда TLS предоставляет байты согласования для передачи, они добавляются в конец байтов согласования для текущего уровня шифрования. После этого уровень шифрования определяет тип пакета, в котором будет передаваться кадр CRYPTO (см. таблицу 1).

Применяются 4 уровня шифрования, создающих ключи для пакетов Initial, 0-RTT, Handshake и 1-RTT. Кадры CRYPTO передаются только на трёх уровнях (без 0-RTT). Этим 4 уровням соответствуют три пространства номеров пакетов - пакеты Initial используют отдельные пространства, а 0-RTT и 1-RTT - пространство данных приложения.

QUIC принимает незащищённое содержимое записей согласования TLS как содержимое кадров CRYPTO. Защита записей TLS не используется в QUIC. Кадры CRYPTO протокол QUIC собирает в пакеты QUIC, для которых применяется защита пакетов QUIC.

Кадры QUIC CRYPTO передают лишь сообщения согласования TLS. Сигналы TLS преобразуются в коды ошибок QUIC CONNECTION_CLOSE (см. параграф 4.8). Данные приложения TLS и другие типы содержимого не могут транспортироваться протоколом QUIC на любом из уровней шифрования и получение их от стека TLS является ошибкой.

При получении пакета QUIC с кадром CRYPTO из сети конечная точка выполняет указанные ниже действия.

- Если пакет использует текущий уровень шифрования приёма TLS, данные во входном потоке упорядочиваются как обычно. Как и в кадрах STREAM, смещение служит для нахождения конкретного места последовательности данных. Если в результате этого процесса становятся доступными новые данные, они доставляются TLS по порядку.
- Если пакет относится к установленному ранее уровню шифрования, ему **недопустимо** содержать данные, которые выходят за пределы полученных ранее в этом потоке. Реализация **должна** считать нарушение этого требования ошибкой соединения типа PROTOCOL_VIOLATION.
- Если пакет относится к новому уровню шифрования, но сохраняется для последующей обработки в TLS. Как только TLS перейдёт к приёму с этого уровня шифрования, сохранённые данные могут быть представлены TLS. Когда TLS предоставляет ключи для более высокого уровня шифрования наличие данных от предыдущего уровня шифрования, не потреблённых TLS, **доено** считаться ошибкой соединения PROTOCOL_VIOLATION.

Каждый раз, когда TLS предоставляет новые данные, запрашиваются новые байты согласования от TLS. Протокол TLS может не предоставить байтов, если полученные сообщения согласования неполны или нет данных для передачи.

Содержимое кадров CRYPTO может обрабатываться TLS постепенно или буферизоваться до полного приёма сообщений. TLS отвечает за буферизацию байтов согласования, полученных с соблюдением порядка. QUIC отвечает за буферизацию байтов согласования, полученных с нарушением порядка или для уровня шифрования, который ещё не готов. QUIC не предоставляет средств управления потоком кадров CRYPTO (см. параграф 7.5 [QUIC-TRANSPORT]).

Завершение согласования TLS указывается протоколу QUIC вместе с финальными байтами согласования, которые TLS требуется передать. На этом этапе проверяется подлинность транспортных параметров, анонсированных партнёром при согласовании (см. параграф 8.2).

По завершении согласования TLS становится пассивным. TLS может продолжать получение данных от своего партнёра и отвечать на них, но передача дополнительных данных не требуется, пока это не будет запрошено

приложением или QUIC. Одной из причин отправки данных является желание сервера предоставить клиенту дополнительные или обновлённые сеансовые квитанции.

По завершении согласования протоколу QUIC нужно лишь представить в TLS любые данные, прибывшие в потоках CRYPTO. Таким же способом, который применялся при согласовании, новые данные запрашиваются у TLS после предоставления принятых данных.

4.1.4. Смена уровня шифрования

Когда ключи данного уровня шифрования становятся доступными TLS, протокол TLS указывает QUIC, что ключи чтения и записи данного режима шифрования доступны. Доступность новых ключей всегда является результатом ввода данных в TLS. Протокол TLS предоставляет новые ключи лишь после инициализации (клиентом) или при предоставлении новых данных согласования.

Однако реализация TLS может выполнять часть обработки асинхронно. В частности, процесс проверки сертификата может занимать некоторое время. В ожидании завершения обработки TLS конечной точке **следует** буферизовать полученные пакеты, если они могут быть обработаны с ключами, которые пока ещё не доступны. Эти пакеты можно будет обработать после предоставления ключей протоколом TLS. Конечной точке **следует** продолжать отвечать на пакеты, которые могут быть обработаны в это время.

После обработки входных данных TLS может создавать байты согласования, ключи для новых уровней шифрования или то и другое вместе. TLS предоставляет протоколу QUIC 3 элемента, когда новый уровень шифрования становится доступным:

- секрет;
- функция аутентифицированного шифрования со связанными данными (Authenticated Encryption with Associated Data или AEAD);
- функция вывода ключа (Key Derivation Function или KDF).

Эти значения основаны на согласуемых TLS значениях и применяются QUIC в Handshake Confirmed для генерации ключей защиты пакета и заголовков (см. раздел 5 и параграф 5.4).

0-RTT (при возможности) готов к использованию после того, как клиент передаст сообщение TLS ClientHello или сервер получит его. После предоставления клиенту QUIC первых байтов согласования стек TLS может сигнализировать смену ключей 0-RTT. На сервере после получения байтов согласования с сообщением ClientHello сервер TLS может сигнализировать доступность ключей 0-RTT.

Хотя TLS в каждый момент использует лишь один уровень шифрования, QUIC может использовать несколько. Например, после отправки сообщения Finished (с использованием кадра CRYPTO на уровне шифрования Handshake) конечная точка может передать данные STREAM (шифрование 1-RTT). Если сообщение Finished теряется, конечная точка использует уровень шифрования Handshake для повторной передачи потерянного сообщения. Нарушение порядка или потеря пакетов могут означать для QUIC необходимость обрабатывать пакеты с разными уровнями шифрования. В процессе согласования это означает возможность обработки пакетов с более высоким и более низким уровнем шифрования по сравнению с текущим уровнем, используемым TLS.

В частности, реализациям сервер нужна возможность чтения пакетов с уровнем шифрования Handshake одновременно с уровнем 0-RTT. Клиент может чередовать кадры ACK, защищённые ключами Handshake и данных 0-RTT, а серверу нужно обрабатывать эти подтверждения для обнаружения потери пакетов Handshake.

Для QUIC нужен также доступ к ключам, которые обычно могут быть недоступны реализации TLS. Например, клиенту может потребоваться подтвердить пакеты Handshake, прежде чем он будет готов передать кадры CRYPTO с этим уровнем шифрования. Поэтому TLS нужно предоставить эти ключи протоколу QUIC, прежде чем он создаст их для своего использования.

4.1.5. Сводка по интерфейсу TLS

На рисунке 5 показан обмен между QUIC и TLS для клиента и сервера. Сплошные стрелки указывают пакеты с данными согласования, пунктирные - пакеты с данными приложения. Каждая стрелка помечена применяемым уровнем шифрования.

На рисунке показано несколько пакетов, образующих одну отправку (flight) сообщений, которые обрабатываются по отдельности, для демонстрации разных действий, вызываемых входящими сообщениями. Это показано несколькими вызовами Get Handshake для извлечения сообщений с разными уровнями шифрования. Новые сообщения согласования запрашиваются после обработки входящих пакетов.

Рисунок 5 иллюстрирует одну из простых структур для одного обмена при согласовании. Точный процесс зависит от структуры реализации конечных точек и порядка прибытия пакетов. Реализации могут использовать иное число операций или выполнять их в другом порядке.

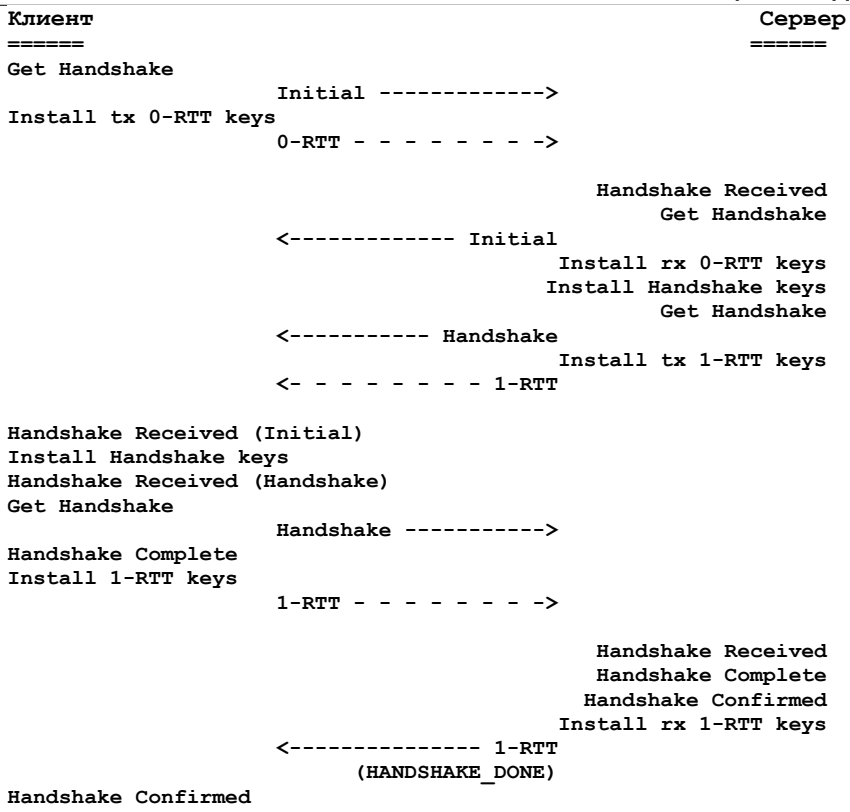

Рисунок 5. Сводка взаимодействия между QUIC и TLS.

4.2. Версия TLS

Этот документ описывает использование TLS 1.3 [TLS13] с QUIC. На практике согласование TLS выбирает версию протокола TLS для применения. Это может приводит к выбору версии новее TLS 1.3, если обе конечных точки поддерживают её. Это приемлемо, если функции TLS 1.3, используемые QUIC, поддерживаются новой версией.

Клиентам **недопустимо** предлагать версии TLS до 1.3. некорректно настроенная реализация TLS может согласовать TLS 1.2 или другую старую версию. Конечная точка **должна** прервать соединение при согласовании версии TLS до 1.3.

4.3. Размер ClientHello

Первый пакет Initial от клиента содержит начало или все первое сообщение криптографического согласования, которым для TLS является ClientHello. Серверам может потребоваться синтаксический анали всего ClientHello (например, для доступа к таким расширениям, как SNI[1] или ALPN[2]) для принятия решения о восприятии нового входящего соединения QUIC. Если сообщение ClientHello разделено между несколькими пакетами Initial, серверу придётся буферизовать полученные в начале фрагменты, что может потребовать избыточных ресурсов, если адрес клиента ещё не проверен. Для предотвращения этого серверы **могут** использовать свойство Retry (см. параграф 8.1 в [QUIC-TRANSPORT]) для буферизации частичных сообщений ClientHello лишь от клиентов с проверенным адресом.

Пакет QUIC и кадрирование добавляют по меньшей мере 36 байтов к размеру сообщения ClientHello. Эти издержки возрастают при выборе клиентом непустого поля Source Connection ID. Издержки не включают маркер и Destination Connection ID длиннее 8 байтов, которые могут потребоваться, если сервер передал пакет Retry.

Типичное сообщение TLS ClientHello может легко поместиться в пакет размером 1200 байтов, однако в дополнение к издержкам QUIC имеется несколько переменных, которые могут вызвать превышение этого размера. Большие сеансовые квитанции, большие или многочисленные общие ключи, длинные списки поддерживаемых шифров, алгоритмов подписи, версий, транспортных параметров QUIC и другие согласуемые параметры и расширения ведут к росту сообщения.

Для серверов, помимо идентификаторов соединения и маркеров, на способность клиента подключиться может влиять размер сеансовых квитанций TLS. Минимизация размера этих параметров повышает вероятность применимости их для клиента и размещения всего сообщения ClientHello в первом пакете Initial.

Реализация TLS не обязана гарантировать, что сообщение ClientHello достаточно велико в соответствии с требованиями QUIC для дейтаграмм с пакетом Initial (см. параграф 14.1 в [QUIC-TRANSPORT]). Реализации QUIC используют кадры PADDING или объединение пакетов для увеличения размера дейтаграмм.

4.4. Проверка подлинности партнёра.

Требования к аутентификации зависят от применяемого прикладного протокола. TLS обеспечивает проверку подлинности сервера и позволяет серверам проверить подлинность клиентов. Клиент **должен** проверять подлинность сервера. Обычно это включает проверку включения идентификации сервера в сертификат и выпуск сертификата доверенной организацией (см. пример в [RFC2818]).

[1]Server Name Identification - идентификация имени сервера.
[2]Application-Layer Protocol Negotiation - согласование протокола прикладного уровня.

Примечание. Если сервер предоставляет сертификаты для аутентификации, размер цепочки сертификатов может быть достаточно большим. Контролирование размера цепочек сертификатов очень важно для производительности QUIC, поскольку серверы могут передавать не более 3 байтов в ответ на каждый полученный байт, пока адрес клиента не проверен (см. параграф 8.1 в [QUIC-TRANSPORT]). Размер цепочки сертификатов можно сократить, ограничивая число имён или расширений, используя компактное представление открытых ключей, подобное ECDSA, или сжимая сертификаты [COMPRESS].

Сервер может запросить проверку подлинности клиента в процессе согласования. Сервер **может** отвергать клиентов, которых невозможно аутентифицировать по запросу. Требования к аутентификации клиента зависят от прикладного протокола и развёртывания.

Серверу **недопустимо** проверять подлинность клиента после согласования (как указано в параграфе 4.6.2 [TLS13]), поскольку мультиплексирование в QUIC не позволяет клиентам сопоставить запрос сертификата с вызвавшим его событием на уровне приложения (см. [HTTP2-TLS13]). Более конкретно, серверам **недопустимо** передавать сообщения TLS CertificateRequest после согласования, а клиенты **должны** считать получение такого сообщения ошибкой соединения типа PROTOCOL_VIOLATION.

4.5. Возобновление сессии

QUIC может использовать функцию восстановления сессии TLS 1.3. Это делается путём передачи сообщений NewSessionTicket в кадрах CRYPTO после завершения согласования. Возобновление сессии моно использовать для обеспечения 0-RTT даже при отключённом 0-RTT.

Конечным точкам, возобновляющим сессию, может потребоваться запомнить некую информацию о текущем соединении при создании возобновлённого соединения. TLS требует сохранения некой информации (см. параграф 4.6.1 в [TLS13]). Протокол QUIC не зависит от сохранения какого-либо состояния при возобновлении соединения, пока не применяется 0-RTT (см. параграф 7.4.1 в [QUIC-TRANSPORT] и параграф 4.6.1). Протокол приложения может зависеть от состояния, сохраняемого между соединениями.

Клиенты могут сохранять состояние, требуемое для возобновления, вместе с сеансовой квитанцией, а серверы могут использовать такую квитанцию для переноса состояния.

Возобновление сессии помогает серверам связывать действия в исходном соединении с возобновляемым, что может создавать для клиентов проблемы приватности. Клиенты могут отказаться от возобновления сессий для предотвращения такого сопоставления. Клиентам **не следует** повторно применять квитанции, поскольку это может помочь сторонним объектам сопоставить соединения (см. Приложение C.4 к [TLS13]).

4.6. 0-RTT

0-RTT в QUIC позволяет клиенту передать данные приложения до того, как будет завершено согласование. Это позволяет повторно использовать согласованные параметры из предыдущего соединения. Для поддержки этого 0-RTT зависит от запоминания клиентом важных параметров и предоставления серверу сеансовой квитанции TLS, позволяющей тому восстановить информацию. Эти сведения включают параметры, определяющие состояние TLS в соответствии с [TLS13], транспортные параметры QUIC, выбранный протокол приложения и любые данные, которые могут быть нужны приложению (см. параграф 4.6.3). Эта информация определяет формирование и содержимое пакетов 0-RTT.

Для обеспечения доступности одних сведений обеим конечным точкам вся информация, используемая для организации 0-RTT, берётся из одного соединения. Конечные точки не могут выборочно игнорировать данные, которые могут влиять на передачу или обработку 0-RTT.

[TLS13] устанавливает 7-дневное ограничение между исходным соединением и попыткой использования 0-RTT. Имеются и другие ограничения для применения 0-RTT, в частности связанные с возможностью replay-атак (см. параграф 9.2).

4.6.1. Включение 0-RTT

Определено расширение TLS early_data в сообщении NewSessionTicket для передачи (в параметре max_early_data_size) объема данных TLS 0-RTT, которые сервер готов воспринять. QUIC не использует ранние данные TLS, применяя для передачи таких данных пакеты 0-RTT. Соответственно, параметр max_early_data_size переназначен для контрольного значения 0xffffffff, указывающего, что сервер готов воспринимать данные QUIC 0-RTT. Отсутствие расширения early_data в NewSessionTicket говорит о нежелании сервера принимать данные 0-RTT. Объем данных, которые клиент может передать в QUIC 0-RTT, задаёт переданный сервером параметр initial_max_data.

Серверам **недопустимо** передавать расширение early_data с полем max_early_data_size, отличным от 0xffffffff. Клиент **должен** считать получение NewSessionTicket с расширением early_data, содержащим иное значение, ошибкой соединения типа PROTOCOL_VIOLATION.

Клиент, желающий передавать пакеты 0-RTT, использует расширение early_data в сообщении ClientHello последующего согласования (см. параграф 4.2.10 в [TLS13]) и затем передаёт данные в пакетах 0-RTT.

Клиент, пытающийся передать 0-RTT, может также предоставить маркер проверки адреса, если сервер передал ему пакет NEW_TOKEN (см. параграф 8.1 в [QUIC-TRANSPORT]).

4.6.2. Восприятие и отклонение 0-RTT

Сервер воспринимает 0-RTT, отправляя расширение early_data в EncrypteEncryptedExtensions (см. параграф 4.2.10 в [TLS13]). После этого сервер обрабатывает и подтверждает получаемые пакеты 0-RTT. Чтобы отвергнуть 0-RTT, сервер передаёт EncryptedExtensions без расширения early_data. Сервер всегда будет отвергать 0-RTT, если он передал TLS HelloRetryRequest. Отвергающему 0-RTT серверу **недопустимо** обрабатывать пакеты 0-RTT, даже когда он способен делать это. Если пакеты 0-RTT отвергнуты, клиенту **следует** считать приём подтверждения пакета 0-RTT ошибкой соединения PROTOCOL_VIOLATION, если он может обнаруживать это условие.

Когда 0-RTT отвергается, все характеристики соединения, предполагаемые клиентом, могут быть некорректными. Это включает выбор прикладного протокола, транспортные параметры и конфигурацию приложения. Поэтому клиент **должен** сбросить состояния всех потоков и связанные с потоками состояния соединений.

Клиент **может** повторить попытку 0-RTT при получении пакета Retry или Version Negotiation. Эти пакеты не означают отказа от 0-RTT.

4.6.3. Проверка конфигурации 0-RTT

При получении сервером ClientHello с расширением early_data он должен принять решение о восприятии или отклонении 0-RTT от клиента. Отчасти решение определяет стек TLS (например, проверка наличия возобновляемого шифра в ClientHello, см. параграф 4.2.10 в [TLS13]). Даже при отсутствии у стека TLS причин отвергать 0-RTT, стек QUIC или прикладной протокол, использующий QUIC, может отклонить данные 0-RTT по причине несовместимости конфигурации транспорта или приложения, связанной с возобновляемым соединением, с текущей настройкой сервера.

QUIC требует связывания дополнительного состояния транспорта с сеансовой квитанцией 0-RTT. Одним из основных способов реализации этого является использование сеансовых квитанций без учёта состояния и сохранение состояния в этой квитанции. Прикладной протокол, использующий QUIC, может иметь похожие требования в части связывания или сохранения статуса. Это связанное состояние применяется при решении вопроса об отклонении 0-RTT. Например, настройки HTTP/3 [QUIC-HTTP] могут определять интерпретацию клиентских данных 0-RTT. Другие приложения, использующие QUIC, могут иметь свои требования в части восприятия или отклонения данных 0-RTT.

4.7. HelloRetryRequest

Сообщение HelloRetryRequest (см. параграф 4.1.4 в [TLS13]) может служить для запроса у клиента новых сведений, таких как общий ключ, или для проверки некоторых характеристик клиента. С точки зрения QUIC сообщение HelloRetryRequest не отличается от других сообщений криптографического согласования, передаваемых в пакетах Initial. Хотя это свойство можно применять для проверки адреса, реализациям QUIC **следует** использовать для этого Retry (см. параграф 8.1 в [QUIC-TRANSPORT]).

4.8. Ошибки TLS

При возникновении ошибки TLS генерирует сигнал в соответствии с разделом 6 в [TLS13], который преобразуется в ошибку соединения QUIC. Значение AlertDescription добавляется к 0x0100 для создания кода ошибки QUIC из диапазона для CRYPTO_ERROR (параграф 20.1 в [QUIC-TRANSPORT]) и полученное значение передаётся в кадре QUIC CONNECTION_CLOSE типа 0x1c.

QUIC может передавать лишь сигналы критического уровня (fatal). В TLS 1.3 единственным применением уровня предупреждения (warning) является сигнализация о закрытии соединения (параграф 6.1 в [TLS13]). QUIC поддерживает другие механизмы прерывания соединений, а соединение TLS закрывается лишь при ошибке и конечная точка QUIC **должна** считать любой сигнал от TLS, как сигнал уровня fatal.

QUIC позволяет использовать созданный код вместо конкретного кода ошибки (см. раздел 11 в [QUIC-TRANSPORT]). Для сигналов TLS это включает замену любого сигнала базовым, таким как handshake_failure (0x0128 в QUIC). Конечные точки **могут** использовать базовый код ошибки для предотвращения возможного раскрытия конфиденциальных сведений.

4.9. Отбрасывание неиспользуемых ключей

После перехода QUIC на новый уровень шифрования ключи предыдущего уровня можно отбросить. Это происходит несколько раз в процессе согласования, а также при обновлении ключей (6. Обновление ключей).

Ключи защиты пакетов не отбрасываются сразу же при появлении новых ключей. Если пакеты из прежнего уровня шифрования содержат кадры CRYPTO, кадры с повтором данных **должны** передаваться с тем же уровнем шифрования. Точно также конечная точка генерирует подтверждения пакетов на том же уровне, что и в подтверждаемых пакетах. Таким образом, ключи более низкого уровня шифрования могут быть нужны в течение короткого времени после появления ключей для более нового уровня шифрования.

Конечная точка не может отбросить ключи для данного уровня шифрования, пока она не получит от партнёра все сообщения криптографического согласования на данном уровне от своего партнёра и партнёр не сделает то же самое. Для ключей Initial (параграф 4.9.1) и Handshake (параграф 4.9.2) предусмотрены другие методы контроля отбрасывания. Эти методы не препятствуют приёму или отправке пакетов на данном уровне шифрования, поскольку партнёр мог не получить все требуемые подтверждения.

Хотя конечная точка может сохранять старые ключи, новые данные **должны** передаваться с наивысшим доступным уровнем шифрования, а прежний уровень можно применять лишь для кадров ACK и данных, повторяемых в кадрах CRYPTO. Такие пакеты **могут** также включать кадры PADDING.

4.9.1. Отбрасывание ключей Initial

Пакеты, защищённые секретами Initial (параграф 5.2), не аутентифицируются, что позволяет злоумышленнику подделать пакеты с намерением нарушить соединение. Для ограничения таких атак ключи защиты пакетов Initial отбрасываются более активно, чем другие ключи.

Успешное использование пакетов Handshake показывает, что обмен пакетами Initial больше не требуется, поскольку нужные для этого ключи могут создаваться лишь после получения всех кадров CRYPTO из пакетов Initial. Поэтому клиент **должен** отбросить ключи Initial при первой отправке пакета Handshake, а сервер **должен** отбросить эти ключи при первой успешной обработке пакета Handshake. После этого конечным точкам **недопустимо** передавать пакеты Initial.

Это ведёт к отказу от состояния восстановления потерь для уровня шифрования Initial и игнорированию оставшихся пакетов Initial.

4.9.2. Отбрасывание ключей Handshake

Конечная точка **должна** отбросить ключи Handshake после подтверждения согласования TLS (параграф 4.1.2).

4.9.3. Отбрасывание ключей 0-RTT

Пакеты 0-RTT и 1-RTT используют общее пространство номеров и клиенты не передают пакеты 0-RTT после отправки 1-RTT (параграф 5.6). Поэтому клиенту **следует** отбрасывать ключи 0-RTT, как только он установит ключи 1-RTT.

Кроме того, сервер **может** отбросить ключи 0-RTT, как только получит пакет 1-RTT. Однако в результате нарушения порядка пакеты 0-RTT могут приходить после 1-RTT. Сервер **может** временно сохранить ключи 0-RTT для обработки доставленных с нарушением порядка пакетов без необходимости повторять их содержимое с ключами 1-RTT. После приёма пакета 1-RTT сервер **должен** отбрасывать пакеты 0-RTT по истечении короткого времени, для которого **рекомендуется** устанавливать утроенное значение тайм-аута зондирования (Probe Timeout или PTO, см. [QUIC-RECOVERY]). Сервер **может** начать отбрасывание 0-RTT раньше, если он понимает, что все пакеты 0-RTT получены, что можно определить путём отслеживания пропущенных номеров.

5. Защита пакета

Как и TLS про протоколу TCP, QUIC защищает пакеты с помощью ключей, выведенных из согласования TLS с использованием алгоритма AEAD [AEAD] согласованного TLS. Защита пакетов QUIC зависит от их типа:

- для пакетов Version Negotiation криптографическая защита не применяется;
- пакеты Retry используют AEAD_AES_128_GCM для защиты от случайного изменения и ограничения числа объектов, которые могут создавать действительные пакеты Retry (см. параграф 5.8);
- пакеты Initial применяют AEAD_AES_128_GCM с ключами, выведенными из поля Destination Connection ID в первом пакете Initial от клиента (см. параграф 5.2);
- для всех остальных пакетов используется строгая криптографическая защита конфиденциальности и целостности с использованием ключей и алгоритмов, согласованных TLS.

В этом разделе описано применение защиты к пакетам Handshake, 0-RTT и 1-RTT. Такая же защита используется для пакетов Initial. Однако определить ключи защиты пакетов Initial несложно, поэтому конфиденциальность и целостность их не считается защищённой. Пакеты Retry используют фиксированный ключ и также не защищены в части конфиденциальности и целостности.

5.1. Ключи защиты пакета

QUIC выводит ключи защиты пакетов таким же способом, как TLS выводит ключи защиты записей.

Каждый уровень шифрования имеет свои секретные значения для защиты пакетов в каждом направлении. Эти секреты трафика выводятся TLS (см. параграф 7.1 и [TLS13]) и применяются QUIC для всех уровней шифрования кроме уровня Initial. Секреты для уровня шифрования Initial рассчитываются на основе исходного значения Destination Connection ID, полученного от клиента, как описано в параграфе 5.2.

Ключи защиты пакетов рассчитываются из секретов TLS с использованием функции KDF, предоставляемой TLS. В TLS 1.3 применяется функция HKDF-Expand-Label, описанная в параграфе 7.1 [TLS13], с хэш-функцией из согласованного шифронабора. При любом применении HKDF-Expand-Label в QUIC используется Context нулевого размера (пустой). Отметим, что метки, описываемые строками, кодируются как байты с использованием кодировки ASCII [ASCII] без кавычек и NUL-байта в конце.

Другие версии TLS **должны** предоставлять похожую функцию для использования в QUIC.

Секрет текущего уровня шифрования и метка quic key служат входными данными KDF для создания ключа AEAD, а метка quic iv служит для вывода вектора инициализации (Initialization Vector или IV, см. параграф 5.3). Ключ защиты заголовка использует метку quic hp (см. параграф 5.4). Применение меток обеспечивает разделение ключей QUIC и TLS (см. параграф 9.6).

Метки quic key и quic hp применяются для создания ключей, поэтому размер Length, предоставляемый HKDF-Expand-Label, вместе с этими метками определяется размером ключей в AEAD или алгоритме защиты заголовка. Length для quic iv является минимальным размером AEAD nonce или составляет 8 байтов, если тот больше [AEAD].

KDF, используемой для начальных секретов, всегда служит функция HKDF-Expand-Label из TLS 1.3 (см. параграф 5.2).

5.2. Секреты Initial

Пакеты Initial используют защиту, но секрет выводится из поля Destination Connection ID в первом пакете Initial от клиента. Этот секрет определяется использованием HKDF-Extract (см. параграф 2.2 в [HKDF]) с затравкой (salt) 0x38762cf7f55934b34d179ae6a4c80cadccbb7f0a и входным ключевым материалом (input keying material или IKM) из поля Destination Connection ID. В результате создаётся промежуточный псевдослучайный ключ (pseudorandom key или PRK), применяемый для создания двух секретов, используемых при передаче и приёме.

Секрет, применяемый клиентами при создании пакетов Initial, использует PRK и метку client in как входные данные функции HKDF-Expand-Label из TLS [TLS13] для создания 32-байтового секрета. В создаваемых сервером пакетах применяется тот же процесс с меткой server in. Хэш-функцией для HKDF при создании начальных секретов служит SHA-256 [SHA]. Псевдокод процесса приведён ниже.

```
initial_salt = 0x38762cf7f55934b34d179ae6a4c80cadccbb7f0a
initial_secret = HKDF-Extract(initial_salt, client_dst_connection_id)

client_initial_secret = HKDF-Expand-Label(initial_secret,
                                          "client in", "",
                                          Hash.length)
server_initial_secret = HKDF-Expand-Label(initial_secret,
```

```
                "server in", "",
                Hash.length)
```

Идентификатор соединения, используемый с HKDF-Expand-Label, - это Destination Connection ID из пакета Initial, переданного клиентом. Это поле содержит случайное значение, если клиент не создаёт пакет Initial после получения пакета Retry, где значение Destination Connection ID выбрано сервером.

Будущим версиям QUIC **следует** применять новое значение затравки, чтобы ключи отличались в разных версиях QUIC. Это не позволит промежуточному устройству, распознающему лишь одну версию QUIC, просматривать или изменять содержимое пакетов будущих версий. Функция HKDF-Expand-Label из TLS 1.3 **должна** применяться для пакетов Initial даже при отсутствии TLS 1.3 среди предлагаемых версий TLS.

Секреты, применяемые для создания последующих пакетов Initial, меняются, когда сервер передает пакет Retry с выбранным им идентификатором соединения. Секреты не меняются при изменении клиентом значения Destination Connection ID, используемого им в ответ на пакет Initial от сервера.

> Примечание. Поле Destination Connection ID может иметь любой размер до 20 байтов, включая 0, если сервер передал пакет Retry с пустым полем Source Connection ID. После Retry ключи Initial не обеспечивают клиенту гарантии получения его пакета сервером, поэтому клиенту приходиться полагаться на обмен, включающий пакет Retry, для проверки адреса сервера (см. параграф 8.1 в [QUIC-TRANSPORT]).

Примеры пакетов Initial представлены в Приложении A.

5.3. Применение AEAD

Функция AEAD [AEAD], применяемая для защиты пакетов QUIC, является AEAD, согласованной для использования в соединении TLS. Например, если TLS использует шифр TLS_AES_128_GCM_SHA256, будет применяться функция AEAD_AES_128_GCM.

QUIC может применять любой шифронабор, заданный в [TLS13], за исключением TLS_AES_128_CCM_8_SHA256. **Недопустимо** согласовывать шифр, для которого не определена схема защиты заголовков. Этот документ определяет схему защиты заголовков для всех шифров, заданных в [TLS13], кроме TLS_AES_128_CCM_8_SHA256. Эти шифры имеют 16-байтовый тег аутентификации и дают на выходе рост на 16 байтов по сравнению со входными данными.

Конечной точке **недопустимо** отвергать сообщение ClientHello, предлагающее неподдерживаемый шифр, поскольку это не позволит развёртывать новые шифронаборы. Это применимо и к шифру TLS_AES_128_CCM_8_SHA256.

При создании пакетов функция AEAD применяется до защиты заголовка (см. параграф 5.4). Незащищённый заголовок является частью связанных данных (associated data или A). При обработке пакетов конечная точка сначала снимает защиту заголовка.

Ключ и IV для пакета рассчитываются в соответствии с параграфом 5.1. Значение nonce (N) формируется путём объединения IV защиты пакета с номером пакета. 62 бита восстановленного номера пакета QUIC в сетевом порядке байтов дополняются слева нулями до размера IV. После этого применяется операция «исключительное ИЛИ» (XOR) к дополненному номеру пакета и IV для создания AEAD.

Связанными данными (A) для AEAD является содержимое заголовка QUIC, начиная с первого байта короткого или длинного заголовка и заканчивая незащищённым номером пакета (включительно). Входными открытыми данными (plaintext или P) для AEAD служит содержимое (payload) пакета QUIC, как описано в [QUIC-TRANSPORT]. Выходной шифротекст (ciphertext или C) функции AEAD передаётся в пакете вместо P.

Некоторые функции AEAD имеют ограничения на число пакетов, шифруемых с одним ключом и IV (см. параграф 6.6). Этот предел может быть меньше предельного числа пакетов. Конечная точка **должна** запустить обновление ключей (раздел 6) до достижения предела, установленного для используемой функции AEAD.

5.4. Защита заголовка

Поля заголовка пакетов QUIC, в частности, Packet Number, защищаются с использованием ключа, выведенного отдельно от ключа защиты пакета и IV. Ключ выводится с использованием метки quic hp и применяется для защиты конфиденциальности полей, не раскрываемых элементам пути доставки.

Эта защита применяется для младших битов первого байта и полю Packet Number. Для пакетов с длинным заголовком защищаются 4 младших бита первого байта, для пакетов с коротким заголовком - 5. Для обоих вариантов заголовка это включает резервные биты и поле Packet Number Length, а для короткого заголовка - бит Key Phase.

В течение срока соединения применяется один ключ защиты заголовков без смены при обновлении ключей (раздел 6). Это позволяет защитить заголовок в Key Phase. Процесс защиты не применяется для пакетов Retry и Version Negotiation, которые не включают защищаемых данных или полей заголовков.

5.4.1. Применение защиты заголовка

Защита заголовка применяется после защиты пакета (см. параграф 5.3). Делается выборка шифрованного содержимого пакета, которая служит вводом для алгоритма шифрования. Алгоритм зависит от согласования AEAD. Выходом алгоритма является 5-байтовая маска, применяемая к защищаемым полям заголовка в операции «исключительное ИЛИ» (XOR). Младшие биты первого байта пакета маскируются младшими битами первого байта маски, а порядковый номер - её. оставшимися байтами. Лишние биты маски (если номер короткий) не применяются.

На рисунке 6 показан пример алгоритма для защиты заголовка. Снятие защиты заголовка отличается лишь порядком определения размера номера пакета (pn_length), ^ представляет операцию XOR.

Конкретные функции защиты заголовка определяются на основе выбранного шифра (см. параграфы 5.4.3 и 5.4.4).

```
mask = header_protection(hp_key, sample)

pn_length = (packet[0] & 0x03) + 1
if (packet[0] & 0x80) == 0x80:
   # Long header: 4 bits masked
   packet[0] ^= mask[0] & 0x0f
else:
   # Short header: 5 bits masked
   packet[0] ^= mask[0] & 0x1f

# pn_offset – начало поля Packet Number.
packet[pn_offset:pn_offset+pn_length] ^= mask[1:1+pn_length]
```

Рисунок 6. Псевдокод защиты заголовка.

На рисунке 7 показан пример пакетов с длинным (Initial) и коротким (1-RTT) заголовком и указаны поля в каждом заголовке, охватываемые защитой, а также выбираемые части защищённого пакета.

```
Initial Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 0,
  Reserved Bits (2),         # Защищено
  Packet Number Length (2),  # Защищено
  Version (32),
  DCID Len (8),
  Destination Connection ID (0..160),
  SCID Len (8),
  Source Connection ID (0..160),
  Token Length (i),
  Token (..),
  Length (i),
  Packet Number (8..32),     # Защищено
  Protected Payload (0..24), # Пропускается
  Protected Payload (128),   # Выбирается
  Protected Payload (..)     # Остальное
}

1-RTT Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),         # Защищено
  Key Phase (1),             # Защищено
  Packet Number Length (2),  # Защищено
  Destination Connection ID (0..160),
  Packet Number (8..32),     # Защищено
  Protected Payload (0..24), # Пропускается
  Protected Payload (128),   # Выбирается
  Protected Payload (..),    # Остальное
}
```

Рисунок 7. Пример защиты заголовка и шифротекста.

До того, как шифр TLS можно будет применять в QUIC, **должен** быть задан алгоритм защиты заголовка и AEAD, используемый с этим шифром. Этот документ задаёт алгоритмы AEAD_AES_128_GCM, AEAD_AES_128_CCM, AEAD_AES_256_GCM (AES AEAD, определённые в [AEAD]) и AEAD_CHACHA20_POLY1305 (определён в [CHACHA]). До выбора шифронабора TLS применяется защита заголовка AES (параграф 5.4.3), соответствующая защите пакета AEAD_AES_128_GCM.

5.4.2. Пример защиты заголовка

Алгоритм защиты заголовка использует ключ защиты заголовка и выборку шифротекста из поля Payload в пакете. Отбирается всегда одинаковое количество данных, но необходимо учитывать снятие защиты на приёмной стороне, которая не знает размер поля Packet Number. Выборка шифрованных данных начинается со смещением 4 байта от начала поля Packet Number, т. е. при выборке предполагается 4-байтовое поле Packet Number (максимальный размер).

Конечная точка **должна** отбрасывать пакеты, размера которых недостаточно для полной выборки. Для обеспечения достаточного размера выборки пакеты дополняются так, суммарный размер кодированного номера пакета и шифрованного содержимого по меньшей мере на 4 байта превышает размер требуемой для защиты заголовка выборки. Шифры, заданные в [TLS13] (кроме TLS_AES_128_CCM_8_SHA256, не применяемого в схеме защиты заголовка) имеют 16-байтовые расширения и 16-байтовую выборку для защиты заголовка. Это требует по меньшей мере 3 байтов в кадрах незащищённого содержимого, если номер пакета кодируется одним байтом или 2 байтов в кадрах при 2-байтовом кодировании номера пакета. Выборку шифротекста можно реализовать приведённым ниже псевдокодом.

```
# pn_offset – начало поля Packet Number.
sample_offset = pn_offset + 4

sample = packet[sample_offset..sample_offset+sample_length]
```

Смещение номера пакета в коротком заголовке определяется как

```
pn_offset = 1 + len(connection_id)
```

В пакете с длинным заголовком смещение порядкового номера рассчитывается как

```
pn_offset = 7 + len(destination_connection_id) +
```

```
                len(source_connection_id) +
                len(payload_length)
if packet_type == Initial:
    pn_offset += len(token_length) +
                 len(token)
```

Например, для пакета с коротким заголовком, 8-байтовым идентификатором соединения и защитой AEAD_AES_128_GCM, выборка будет включать байты с 13 по 28 (отсчёт от 0). В одну дейтаграмму UDP может включаться несколько пакетов QUIC, каждый из которых обрабатывается отдельно.

5.4.3. Защита заголовка на основе AES

В этом параграфе определён алгоритм защиты пакетов для AEAD_AES_128_GCM, AEAD_AES_128_CCM и AEAD_AES_256_GCM. AEAD_AES_128_GCM и AEAD_AES_128_CCM используют 128-битовый шифр AES в режиме ECB (Electronic Codebook — электронный шифровальный блокнот). AEAD_AES_256_GCM использует 256-битовый шифр AES в режиме ECB. Протокол AES определён в [AES].

Этот алгоритм извлекает 16 шифрованного содержимого и использует в AES-ECB. Псевдокод функции защиты заголовков имеет вид

```
header_protection(hp_key, sample):
  mask = AES-ECB(hp_key, sample)
```

5.4.4. Защита заголовка на основе ChaCha20

При использовании AEAD_CHACHA20_POLY1305 для защиты заголовков применяется raw-функция ChaCha20, определённая в параграфе 2.4 [CHACHA]. Она использует 256-битовый ключ 16-байтовую выборку из защищённой части пакета. Первые 4 байта выборки служат счётчиком блоков. Реализация ChaCha20 может принимать 32-битовое целое число вместо последовательности байтов и в этом случае последовательность байтов интерпретируется как значение little-endian. Оставшиеся 12 используются как nonce. реализация ChaCha20 может принимать массив из трёх 32-битовых целых чисел вместо последовательности байтов и в этом случае байты nonce интерпретируются как последовательность 32-битовых целых чисел little-endian.

Маска шифрования создаётся вызовом ChaCha20 для защиты 5 байтов со значением 0. Псевдокод защиты заголовка имеет вид

```
header_protection(hp_key, sample):
  counter = sample[0..3]
  nonce = sample[4..15]
  mask = ChaCha20(hp_key, counter, nonce, {0,0,0,0,0})
```

5.5. Приём защищённых пакетов

При успешной получении конечной точкой пакета с данным номером она **должна** отбрасывать все пакеты из того же пространства с большими номерами, если не может снять их защиту с тем же (или новым в случае обновления) ключом (см. раздел 6). Точно так же **должны** отбрасываться пакеты, представляющиеся триггерами обновления ключей, если для них не удаётся снять защиту.

Отказ при снятии защиты пакета не обязательно указывает наличие протокольной ошибки или атаки. Отсечка кодирования порядкового номера в QUIC может приводить к некорректному декодированию номеров в случае значительной задержки.

5.6. Использование ключей 0-RTT

Если доступны ключи 0-RTT (см. параграф 4.6.1), отсутствие защиты от повторного использования (replay) означает, что ограничения на применение ключей требуются для защиты от replay-атак на протокол. Среди определённых в [QUIC-TRANSPORT] кадров, для кадров STREAM, RESET_STREAM, STOP_SENDING, CONNECTION_CLOSE использование с 0-RTT может быть небезопасным, поскольку эти кадры содержат данные приложения. Полученные в 0-RTT данные могут заставить приложение на сервере обрабатывать данные несколько раз вместо одного. Предпринимаемые в результате сервером дополнительные действия по обработке повторно использованных данных приложения могут приводить к нежелательным последствиям. Поэтому клиенту **недопустимо** использовать 0-RTT для данных приложения, если это не запрашивается приложением специально.

Использующий QUIC прикладной протокол **должен** включать профиль, определяющий допустимость использования 0-RTT, иначе 0-RTT можно применять лишь для кадров QUIC, не содержащих данных приложения. Например, профиль для HTTP описан в [HTTP-REPLAY] и применяется для HTTP/3 (см. параграф 10.9 в [QUIC-HTTP]).

Хотя повторное использование пакетов может приводить к дополнительным попыткам соединения, влияние обработки воспроизводимых пакетов без данных приложения ограничено сменой состояния затронутого соединения. Согласование TLS не может быть завершено успешно с использованием воспроизводимых пакетов.

Клиент **может** внести дополнительные ограничения для передачи данных до завершения согласования TLS.

В остальном клиент считает ключи 0-RTT эквивалентом ключей 1-RTT, за исключением невозможности передачи некоторых кадров с ключами 0-RTT (см. параграф 12.5 в [QUIC-TRANSPORT]).

Клиент, получивший индикацию восприятия его данных 0-RTT сервером может передавать такие данные, пока не получит от сервера все сообщения процесса согласования (handshake). Клиенту **следует** прекратить передачу данных 0-RTT при получении сведений о том, что данные 0-RTT были отвергнуты.

Серверу **недопустимо** использовать ключи 0-RTT для защиты пакетов и для защиты подтверждений пакетов 0-RTT он использует ключи 1-RTT. Клиенту **недопустимо** пытаться расшифровывать полученные пакеты 0-RTT, он просто **должен** отбрасывать их. После установки клиентом ключей 1-RTT ему **недопустимо** передавать пакеты 0-RTT.

> Примечание. Данные 0-RTT могут быть подтверждены сервером при их получении, но для пакетов с подтверждениями данных 0-RTT клиент не сможет снять защиту до завершения согласования TLS. Для снятия

защиты нужны ключи 1-RTT, которые не могут быть получены раньше приёма от сервера всех сообщений процедуры согласования.

5.7. Получение разупорядоченных защищённых пакетов

В результате нарушения порядка и потерь защищённые пакеты могут быть получены конечной точкой до приёма финальных сообщений согласования TLS. Клиент не сможет расшифровать пакеты 1-RTT от сервера, тогда как сервер может расшифровать пакеты 1-RTT от клиента. Конечным точкам в любой роли **недопустимо** расшифровывать пакеты 1-RTT от партнёра до завершения согласования. Хотя ключи 1-RTT доступны серверу после приёма первых сообщений согласования от клиента, у него нет гарантии состояния клиента.

- Подлинность клиента не проверена, пока сервер не решил использовать распространённый заранее ключ и не проверил привязку заранее распространенного ключа клиента (см. параграф 4.2.11 в [TLS13]).
- Клиент не продемонстрировал работоспособность, пока сервер не проверил адрес клиента с помощью пакета Retry или иным способом (см. параграф 8.1 в [QUIC-TRANSPORT]).
- Все полученные данные 0-RTT, на которые сервер отвечает, могут быть использованными повторно (replay).

Поэтому использование сервером ключей 1-RTT до завершения согласования ограничено отправкой данных. Серверу **недопустимо** обрабатывать входящие пакеты с защитой 1-RTT до завершения согласования TLS. Поскольку отправка подтверждения указывает, что все кадры из пакета обработаны, сервер не может передавать подтверждения для пакетов 1-RTT, пока согласование TLS не завершено. Полученные пакеты с защитой 1-RTT могут сохраняться и расшифровываться по завершении согласования.

Примечание. Реализации TLS могут предоставлять все секреты 1-RTT до завершения согласования. Даже при наличии у реализации QUIC ключей чтения 1-RTT эти ключи не применяются до завершения согласования.

Требование к серверу ждать сообщения Finished от клиента создаёт зависимость от доставки этого сообщения. Клиент может предотвратить возможную блокировку head-of-line, которую это подразумевает, передавая свои пакеты 1-RTT объединёнными с пакетом Handshake, содержащим копию кадра CRYPTO с сообщением Finished, пока один из пакетов Handshake не будет подтверждён. Это позволяет серверу незамедлительно обработать такие пакеты.

Сервер может получить пакеты, защищённые ключами 0-RTT, до приёма сообщения TLS ClientHello. Сервер **может** сохранить эти пакеты для расшифровки после приёма ClientHello.

Клиент обычно получает ключи 1-RTT одновременно с завершением согласования. Даже при наличии секретов 1-RTT клиенту **недопустимо** обрабатывать пакеты с защитой 1-RTT, пока согласование TLS не завершено.

5.8. Целостность пакета Retry

Пакеты Retry (см. параграф 17.2.5 в [QUIC-TRANSPORT]) переносят Retry Integrity Tag, позволяющий отбрасывать пакеты, случайно повреждённые в сети и разрешающий передачу пакета Retry лишь объекту, имеющему доступ к пакету Initial.

Retry Integrity Tag - это 128-битовое поле, возвращаемое функцией AEAD_AES_128_GCM [AEAD], по входным данным:

- секретный ключ K - 128 битов со значением 0xbe0c690b9f66575a1d766b54e368c84e;
- одноразовое значение (nonce) N - 96 битов со значением 0x461599d35d632bf2239825bb;
- нешифрованные данные (plaintext) P с пустым значением;
- связанные данные A, содержащие Retry Pseudo-Packet, как показано на рисунке 8.

Секретный ключ и nonce выводятся вызовом HKDF-Expand-Label с использованием секрета 0xd9c9943e6101fd200021506bcc02814c73030f25c79d71ce876eca876e6fca8e и метками quic key и quic iv (параграф 5.1).

```
Retry Pseudo-Packet {
  ODCID Length (8),
  Original Destination Connection ID (0..160),
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 3,
  Unused (4),
  Version (32),
  DCID Len (8),
  Destination Connection ID (0..160),
  SCID Len (8),
  Source Connection ID (0..160),
  Retry Token (..),
}
```

Рисунок 8. Псевдопакет Retry.

Псевдопакет Retry не передаётся в линию. Он рассчитывается по принятому пакету Retry путём удаления Retry Integrity Tag и добавления в начало двух полей.

ODCID Length

Размер поля Original Destination Connection ID (в байтах), представленный 8-битовым целым числом без знака.

Original Destination Connection ID

Значение поля Destination Connection ID из пакета Initial, вызвавшего данный пакет Retry. Размер этого поля указан в ODCID Length. Наличие этого поля гарантирует возможность передачи действительного пакета Retry лишь объектом, которому доступен пакет Initial.

6. Обновление ключей

Когда согласование подтверждено (см. параграф 4.1.2), конечная точка **может** инициировать обновление ключей.

Бит Key Phase указывает ключи, применяемые для защиты пакетов. Бит Key Phase исходно сброшен (0) для первого набора пакетов 1-RTT и переключается для сигнализации каждого последующего обновления ключей.

Бит Key Phase позволяет получателю обнаружить смену ключевого материала без необходимости получения первого пакета, инициировавшего смену. Конечная точка, заметившая изменение бита Key Phase, обновляет ключи и расшифровывает пакет с измененным значением.

Инициирование обновления ключей ведёт к обновлению ключей в обеих конечных точках. Это отличается от TLS, где конечные точки могут обновляться независимо. Этот механизм заменяет механизм обновления ключей TLS, основанный на сообщениях KeyUpdate, передаваемых с использованием ключей шифрования 1-RTT. Конечным точкам **недопустимо** передавать сообщения TLS KeyUpdate и они **должны** считать получение такого сообщения ошибкой соединения типа 0x010a, эквивалентной сигналы TLS unexpected_message (см. параграф 4.8).

На рисунке 9 показан процесс обмена ключами, где исходные ключи (@M) заменяются новыми (@N). Значение бита Key Phase указано в квадратных скобках.

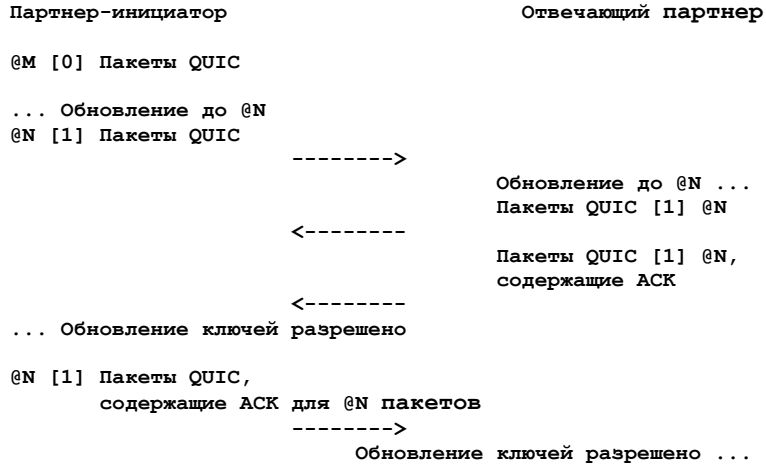

Рисунок 9. Обновление ключа.

6.1. Запуск обновления ключей

Конечные точки поддерживают разные секреты чтения и записи для защиты пакетов. Конечная точка запускает обновление ключей, обновляя свой секрет записи для защиты пакетов и используя его с новыми пакетами. Новый секрет записи создаётся из имеющегося в соответствии с параграфом 7.2 в [TLS13]. При этом используется функция KDF, обеспечиваемая TLS, с меткой quic ku. Соответствующий ключ и IV создаются из этого секрета, как указано в параграфе 5.1. Ключ защиты заголовков не обновляется.

Например, для обновления ключей записи с TLS 1.3 применяется HKDF-Expand-Label

```
secret_<n+1> = HKDF-Expand-Label(secret_<n>, "quic ku", "", Hash.length)
```

Конечная точка меняет бит Key Phase и использует обновленный ключ и IV для защиты последующих пакетов.

Конечной точке **недопустимо** запускать обновление ключа до подтверждения согласования (параграф 4.1.2). Конечной точке **недопустимо** инициировать последующие обновления ключа, пока она не получила подтверждения пакетов, защищённых с ключом из текущей фазы. Это гарантирует доступность ключей обоим партнёрам, прежде чем будет начато обновление. Это можно реализовать путём отслеживания наименьшего номера пакета, переданного с каждой фазой ключа, и наибольшего номера подтверждения в пространстве 1-RTT - как только номер подтверждения станет не меньше наименьшего номера переданного пакета, можно запускать обновления ключа.

> Примечание. Ключи для пакетов, отличных от 1-RTT, не обновляются и выводятся исключительно из статуса согласования TLS.

Конечная точка, запустившая обновление ключей, обновляет также ключи, применяемые для полученных пакетов. Эти ключи будут нужны для обработки пакетов, которые партнёр передаст после обновления.

Конечная точка **должна** сохранять старые ключи, пока не снимет защиту с пакета, отправленного уже с использованием новых ключей. Конечной точке **следует** сохранять старые ключи в течение некоторого времени после снятия защиты с пакета, переданного с использованием новых ключей. Преждевременное отбрасывание ключей партнёр сочтёт потерей пакета, что может оказать негативное влияние на производительность.

6.2. Отклик на обновление ключей

Партнёру разрешено запускать обновление ключей после подтверждения пакета а текущей фазе ключей. Конечная точка обнаруживает обновление ключей, когда обрабатывает пакет с фазой ключа, отличающейся от значения, использованного для защиты последнего пакета, который она передала. Для обработки такого пакета конечная точка применяет следующий ключ защиты пакета и IV. Генерация ключей описана в параграфе 6.3.

если пакет успешно обработан с использованием следующего ключа и IV, партнёр запускает обновление ключей. Конечная точка **должна** обновить свои ключи передачи до соответствующей фазы ключей в отклике, как описано в параграфе 6.1. Ключи передачи **должны** быть обновлены до отправки подтверждения для пакета, принятого с обновлёнными ключами. Подтверждая пакет, вызвавший обновление ключей с защитой обновлёнными ключами, конечная точка сообщает о завершении обновления ключей.

Конечная точка может отложить передачу пакета или подтверждения в соответствии со своим обычным поведением и нет необходимости незамедлительно создавать пакет в ответ на обновление ключа. Следующий пакет, переданный конечной точкой, будет использовать обновлённые ключи. Следующий пакет с подтверждением вызовет завершение обновления ключей. Если конечная точка обнаруживает второе обновление до передачи ею какого-либо пакета с

обновлёнными ключами, это указывает, что партнёр обновил ключи дважды, не дожидаясь подтверждения. Конечная точка **может** считать такое обновление ошибкой соединения типа KEY_UPDATE_ERROR.

Конечная точка, получившая подтверждение в пакете, защищённом старыми ключами, для пакетов, защищённых новыми ключами, **может** считать это ошибкой соединения типа KEY_UPDATE_ERROR. Это показывает, что партнёр получил и подтвердил пакет, вызывающий обновление ключей, но не обновил свои ключи.

6.3. Синхронизация создания ключа приёма

Конечным точкам, отвечающим на кажущееся обновление ключа, недопустимо создавать сигнал побочного канала синхронизации, который может указать, что бит Key Phase был недействителен (см. параграф 9.5). Конечные точки могут использовать случайные ключи защиты пакетов вместо отброшенных ключей, когда обновление ключей ещё не разрешено. Применение случайных ключей гарантирует, что попытки снять защиту пакетов не приведут к изменению синхронизации, а обеспечат отклонение пакетов с недействительным битом Key Phase.

Процесс создания новых ключей защиты пакетов для принимаемых пакетов может показать, что произошло обновление ключей. Конечная точка **может** создать новые ключи в процессе обработки пакета, но это открывает сигнал синхронизации, которым злоумышленник может воспользоваться для определения момента смены ключей и утечки значения бита Key Phase.

Обычно считается, что конечные точки имеют текущие и следующие ключи защиты для принимаемых пакетов. В течение короткого времени после обновления ключей (вплоть до PTO) конечные точки **могут** отложить создание нового набора ключей защиты для принимаемых пакетов. Это позволяет конечным точкам сохранять лишь два комплекта приёмных ключей (см. параграф 6.5).

После генерации следующий набор ключей защиты пакетов **следует** сохранить, даже если полученный пакет был потом отброшен. Пакеты, представляющиеся триггером обновления ключей, легко подделать и хотя процесс обновления ключей не требует значительных усилий, запуск этого процесса можно использовать для DoS-атак. По этой причине конечные точки **должны** быть способны сохранять два набора ключей защиты для принимаемых пакетов (текущий и следующий). Сохранение предыдущего набора ключей не требуется, но может поднять производительность.

6.4. Передача с обновлёнными ключами

Конечные точки никогда не передают пакетов, защищённых старыми ключами, используя лишь текущие ключи. Ключи для защиты пакетов могут отбрасываться сразу после перехода на новые ключи. Пакеты с большими порядковыми номерами **должны** быть защищены с теми же или более новыми ключами, чем пакеты с меньшими номерами. Конечная точка, успешно снявшая защиту со старыми ключами, когда для пакетов с меньшими номерами были использованы новые ключи, **должна** считать это ошибкой соединения типа KEY_UPDATE_ERROR.

6.5. Приём с другими ключами

В процессе обновления ключей могут поступать задержанные в сети пакеты, защищённые старыми ключами. Сохранение старых ключей защиты пакетов позволяет успешно обработать задержанные пакеты.

Поскольку пакеты, защищённые ключами из следующей фазы используют то же значение Key Phase, что и пакеты из следующей фазы, необходимо их различать, если требуется обработка пакетов, защищённых со старыми ключами. Это можно сделать по номерам пакетов. Восстановленный номер, который меньше любого из номеров текущей фазы ключей, использует прежние ключи защиты, а номер, который больше номера любого из пакетов текущей фазы, требует применения ключей защиты пакетов из следующей фазы. Нужны меры предосторожности, чтобы процесс выбора между прежними, текущими и будущими ключами защиты пакетов не создавал побочного канала синхронизации, который можно использовать для снятия защиты пакетов (см. параграф 9.5).

В качестве варианта конечные точки могут сохранять лишь два набора ключей защиты пакетов, переходя с прежнего к следующему по истечении достаточно большого времени, позволяющего учесть нарушение порядка в сети. В этом случае бит Key Phase позволяет однозначно выбрать ключи. Конечная точка **может** внести интервал приблизительно равный интервалу зондирования (Probe Timeout или PTO, см [QUIC-RECOVERY]) после того, как следующий набор ключей становится текущим, до создания следующего набора ключей защиты пакетов. Эти обновлённые ключи **могут** заменить прежние ключи в любой момент. С учётом того, что PTO является субъективной мерой (т. е. у партнёра может быть иное представление о RTT), предполагается, что этого времени будет достаточно, чтобы все разупорядоченные пакеты были сочтены партнёром потерянными, даже если они были подтверждены, и интервал будет достаточно коротким, чтобы партнёр мог запустить дальнейшее обновление ключей.

Конечные точки должны учитывать, что партнёру не удастся расшифровать пакеты, которые вызывают обновление ключей, в течение времени, когда он сохраняет старые ключи. Конечным точкам **следует** выждать 3 интервала PTO перед запуском обновления ключей после приёма подтверждения, указывающего получения предшествующего обновления ключей. Если этого не делать, пакеты могут быть отброшены.

Конечной точке **следует** сохранять старые ключи не более трёх интервалов PTO после приёма пакета, защищённого с использованием новых ключей. По истечении этого времени старые ключи и соответствующие секреты **следует** отбросить.

6.6. Ограничения применимости AEAD

Этот документ задаёт ограничения на применение алгоритмов AEAD, чтобы гарантированно не дать злоумышленникам непропорциональных преимуществ для организации атак на конфиденциальность и целостность коммуникаций QUIC.

Пределы использования, заданные в TLS 1.3, применяются для защиты от атак на конфиденциальность при использовании защиты AEAD. Защита целостности при аутентифицированном шифровании также зависит от ограничения числа попыток подделки пакетов. TLS обеспечивает это путём закрытия соединений, если какая-либо запись не прошла проверку подлинности. QUIC игнорирует не прошедшие аутентификацию пакеты, разрешая множество попыток подделки.

В QUIC ограничения для AEAD в части конфиденциальности и целостности применяются раздельно. Ограничение в части конфиденциальности применяется к числу пакетов, шифруемых с данным ключом. Для целостности ограничение применяется к числу пакетов, расшифрованных в данном соединении. Детали применения этих ограничений для каждого алгоритма AEAD рассмотрены ниже.

Конечные точки **должны** считать число зашифрованных пакетов для каждого набора ключей. Если число пакетов, зашифрованных с одним ключом, превышает предел конфиденциальности для выбранного алгоритма AEAD, конечная точка **должна** прекратить применение этих ключей. Конечные точки **должны** запускать обновление ключей до передачи числа пакетов, превышающего установленный для AEAD предел. Если обновление ключей невозможно или достигнут предел в части целостности, конечная точка **должна** прекратить использование соединения и передавать лишь сброс без учёта состояния в ответ на получение пакетов. Конечным точкам **рекомендуется** незамедлительно закрывать соединение с ошибкой типа AEAD_LIMIT_REACHED до достижения состояния, в котором невозможно обновить ключи.

Для AEAD_AES_128_GCM и AEAD_AES_256_GCM предел в части конфиденциальности составляет 2^{23} зашифрованных пакетов (см. Приложение B.1), для AEAD_CHACHA20_POLY1305 предел превышает возможное число пакетов (2^{62}) и им можно пренебречь, а для AEAD_AES_128_CCM предел составляет $2^{21,5}$ зашифрованных пакетов (см. Приложение B.2). Применение ограничений снижает вероятность того, что атакующий сможет определить применяемый алгоритм AEAD от случайного изменения (см. [AEBounds], [ROBUST], [GCM-MU]).

В дополнение к учёту переданных пакетов конечные точки **должны** считать число принятых пакетов, которые не прошли аутентификацию в течение работы соединения. Если общее число принятых пакетов, не прошедших проверку подлинности (для всех ключей) превышает предел, заданный в части целостности для применяемого алгоритма AEAD, конечная точка **должна** незамедлительно закрыть соединение с ошибкой типа AEAD_LIMIT_REACHED и прекратить обработку последующих пакетов.

Для AEAD_AES_128_GCM и AEAD_AES_256_GCM предел в части целостности составляет 2^{52} недействительных пакетов (см. Приложение B.1), для AEAD_CHACHA20_POLY1305 - 2^{36} недействительных пакетов (см. [AEBounds]), а для AEAD_AES_128_CCM - $2^{21,5}$ недействительных пакетов (см. Приложение B.2). Применение этих ограничений снижает вероятность успешной подделки пакетов злоумышленником (см. [AEBounds], [ROBUST], [GCM-MU]).

Конечные точки, ограничивающие размер пакетов, **могут** устанавливать более высокие пределы в части конфиденциальности и целостности (см. Приложение B). Будущий анализ и спецификации **могут** смягчить требования к ограничениям для AEAD.

Любой шифронабор TLS, указанный для применения с QUIC, **должен** задавать пределы использования соответствующей функции AEAD для сохранения конфиденциальности и целостности. Т. е. **должны** быть заданы пределы для числа пакетов, которые могут быть аутентифицированы, и числа пакетов с отказом при аутентификации. Предоставление ссылки на анализ, обосновывающий представленные значения и использованные при этом анализе допущения, позволяет приспособить ограничения к изменяющимся условиям применения.

6.7. Код ошибки при обновлении ключей

Код ошибки KEY_UPDATE_ERROR (0x0e) служит для сигнализации ошибок, связанных с обновлением ключей.

7. Защита пакетов Initial

Пакеты Initial не могут быть защищены с использованием секретного ключа, поэтому атакующий способен изменить их. QUIC обеспечивает защиту от злоумышленников, не способных читать пакеты, но не пытается предоставить дополнительную защиту от атак с возможностью наблюдения и внедрения пакетов. Некоторые из форм вмешательства, такие как изменение самих сообщений TLS, могут быть замечены, но другие, например, изменение ACK, остаются не обнаруженными. Например, атакующий может внедрить пакет с кадром ACK чтобы создать впечатление о том, что кадр не был получен или представить ложное состояние соединения (например, путём изменения ACK Delay). Отметим, что такие пакеты могут вызывать отбрасывание легитимных пакетов как дубликатов. Реализациям **следует** проявлять осторожность, полагаясь на данные из пакетов Initial, которые не аутентифицированы иным способом.

Злоумышленник также может подделать данные в пакетах Handshake, но такое вмешательство требует изменения сообщений согласования TLS, приводящего к отказу этого согласования.

8. Связанные с QUIC настройки TLS Handshake

Некоторые аспекты согласования TLS меняются при использовании с QUIC. Кроме криптографических параметров в процессе согласования передаются и аутентифицируются параметры транспорта QUIC.

8.1. Согласование протокола

QUIC требует, чтобы криптографическое согласование обеспечивало аутентифицированное согласование протокола. TLS использует согласование протокола прикладного уровня [ALPN] для выбора протокола приложения. Если не задан иной механизм согласования такого протокола, конечные точки **должны** применять ALPN. При использовании ALPN конечные точки **должны** немедленно закрывать соединение (см. параграф 10.2 в [QUIC-TRANSPORT]) с сигналом TLS no_application_protocol (код ошибки QUIC 0x0178, см. параграф 4.8), если прикладной протокол не согласован. Хотя [ALPN] лишь указывает, что серверы используют этот сигнал, клиенты QUIC **должны** по коду 0x0178 прерывать соединение в случае отказа при согласовании ALPN.

Прикладной протокол **может** ограничивать версии QUIC, с которыми он работает. Серверы **должны** выбирать протокол приложений, совместимый с выбранной клиентом версией QUIC. Сервер **должен** считать неспособность выбрать совместимый протокол ошибкой соединения типа 0x0178 (no_application_protocol). Клиент **должен** считать выбор серверов несовместимого прикладного протокола ошибкой соединения типа 0x0178.

8.2. Расширение для транспортных параметров QUIC

Транспортные параметры QUIC передаются в расширении TLS. Разные версии QUIC могут определять различные методы согласования транспортной конфигурации. Включение параметров транспорта в согласование TLS обеспечивает защиту целостности этих параметров.

```
enum {
   quic_transport_parameters(0x39), (65535)
} ExtensionType;
```

Поле extension_data field в расширении quic_transport_parameters содержит значение, определённое используемой версией QUIC.

Расширение quic_transport_parameters передаётся в сообщениях ClientHello и EncryptedExtensions в процессе согласования. Конечные точки **должны** передавать расширение quic_transport_parameters, а конечные точки, получившие сообщение ClientHello или EncryptedExtensions без quic_transport_parameters, **должны** закрывать соединение с ошибкой типа 0x016d (эквивалент критического сигнала TLS missing_extension, см. параграф 4.8).

Транспортные параметры становятся доступными до завершения согласования и сервер может использовать эти значения. Однако транспортные параметры не будут аутентифицированы до завершения согласования, поэтому на их подлинность полагаться нельзя. Любое вмешательство в параметры транспорта приведёт к отказу согласования.

Конечным точкам **недопустимо** передавать это расширение в соединениях TLS, не используемых протоколом QUIC (например, при использовании TLS с TCP [TLS13]). Поддерживающая расширение реализация **должна** передавать критический сигнал unsupported_extension при получении этого расширения для транспорта, отличного от QUIC.

Согласование расширения quic_transport_parameters вызывает исключение EndOfEarlyData (см. параграф 8.3).

8.3. Удаление сообщения EndOfEarlyData

Сообщение TLS EndOfEarlyData не используется с QUIC и протокол не полагается на это сообщение для маркировки завершения данных 0-RTT или сигнализации о смене ключей Handshake. Клиентам **недопустимо** передавать сообщение EndOfEarlyData, а сервер **должен** считать получение кадра CRYPTO в пакете 0-RTT ошибкой соединения типа PROTOCOL_VIOLATION.

В результате сообщение EndOfEarlyData не присутствует в согласовании TLS.

8.4. Запрет режима TLS Middlebox Compatibility

В Приложении D.4 [TLS13] описано изменение согласования TLS 1.3 для обхода ошибок некоторых промежуточных устройств. Режим совместимости с промежуточными устройствами в TLS 1.3 включает установку для поля legacy_session_id 32-байтового значения в ClientHello и ServerHello, затем передачу изменённой записи change_cipher_spec. Оба поля и запись не несут семантического содержимого и игнорируются.

Этот режим не используется в QUIC поскольку применим лишь к промежуточным устройствам, нарушающим работу TLS по протоколу TCP. QUIC также не предоставляет средств доставки записи change_cipher_spec. Клиентам **недопустимо** запрашивать использование режима совместимости TLS 1.3, а серверам **следует** считать получение TLS ClientHello с непустым полем legacy_session_id ошибкой соединения типа PROTOCOL_VIOLATION.

9. Вопросы безопасности

Все вопросы безопасности, связанные с TLS применимы и к использованию TLS в QUIC. Чтение [TLS13] и приложений к нему обеспечивает наилучший способ разобраться с вопросами безопасности QUIC. В этом разделе приведена сводка некоторых важных аспектов безопасности, относящихся к интеграции TLS, хотя многие детали, связанные с безопасностью рассмотрены в оставшейся части документа.

9.1. Связывание сессий

Использование сеансовых квитанций TLS позволяет серверами и возможно другим объектам сопоставлять соединения одного клиента (см. параграф 4.5).

9.2. Replay-атаки с 0-RTT

Как описано в разделе 8 [TLS13], использование ранних данных в TLS открывает возможность для атак с повторным использованием (replay). Использование 0-RTT в QUIC также уязвимо для replay-атак. Конечные точки **должны** реализовать и применять защиту от повторного использования, описанную в [TLS13], однако известно, что эта защита несовершенна, поэтому требуется дополнительное рассмотрение риска от повторного использования.

Протокол QUIC не уязвим для replay-атак за исключением данных о прикладном протоколе, которые он может нести. Управление состоянием протокола QUIC основано на типах кадров, определённых в [QUIC-TRANSPORT], и не подвержено replay-атакам. Обработка кадров QUIC идемпотентна и не может приводить к недействительным состояниям соединения в случае повторного использования, разупорядочения или потери кадров. Соединения QUIC не создают эффектов, распространяющихся за пределы срока действия соединения, за исключением тех, которые создаёт протокол приложения, обслуживаемый QUIC.

Сеансовые квитанции TLS и маркеры проверки адресов служат для передачи конфигурационных данных QUIC между соединениями, в частности, чтобы позволить серверу эффективно восстанавливать состояние, используемое при организации соединения и проверке адреса. Это **недопустимо** применять для передачи семантики приложения между взаимодействующими конечными точками и клиенты **должны** считать эти параметры необрабатываемыми (opaque) значениями. Возможность повторного использования маркеров означает требование строгой защиты для них.

На сервере, воспринимающем 0-RTT в соединении требуется больше затрат, чем в соединениях без 0-RTT. Это включает большие расходы на обработку и вычисления. Серверам, воспринимающим 0-RTT, необходимо учитывать вероятность повторного использования и связанные с этим расходы.

В конечном итоге ответственность за контроль рисков replay-атак с 0-RTT ложится на протокол приложения. Использующий QUIC прикладной протокол **должен** описывать применение 0-RTT и меры, реализуемые для защиты от replay-атак. Анализ связанных с этим рисков должен учитывать все свойства протокола QUIC, связанные с семантикой приложения.

Полное отключение 0-RTT является наиболее эффективным способом защиты от replay-атак.

Расширения QUIC **должны** описывать влияние replay-атак на их работу или запрещать применение расширения в 0-RTT. Прикладной протокол **должен** запрещать использование расширений, передающих семантику приложения в 0-RTT, или обеспечивать стратегию предотвращения повторного использования.

9.3. Ослабление атак с отражением пакетов

Небольшое сообщение ClientHello, вызывающее большой блок сообщений согласования от сервера, можно использовать для атак с отражением, усиливающих трафик от атакующего. QUIC включает три средства защиты от таких атак. Во-первых, клиент **должен** дополнять пакет с ClientHello до минимального размера. Во-вторых, при отклике по неизвестным адресам отправителей серверу запрещено передавать более троекратного объёма полученных данных (см. параграф 8.1 в [QUIC-TRANSPORT]). В-третьих, поскольку подтверждения пакетов Handshake аутентифицируются, при атаке вслепую их нельзя подделать. Эти меры ограничивают степень усиления трафика.

9.4. Анализ защиты заголовков

В [NAN] анализируются алгоритмы аутентифицированного шифрования, обеспечивающие конфиденциальность nonce, называемую преобразованием HN (Hide Nonce). Базовая конструкция защиты заголовков в этом документе основана на одном из таких алгоритмов (HN1). Защита заголовка применяется после защиты AEAD для пакета путём выборки набора байтов (sample) из вывода AEAD и шифрования поля заголовка с применением псевдослучайной функции PRF

```
protected_field = field XOR PRF(hp_key, sample)
```

Варианты защиты заголовка в этом документе используют псевдослучайную перестановку (pseudorandom permutation или PRP) вместо базовой PRF. Однако, поскольку все PRP являются также PRF [IMC], эти варианты не отклоняются от конструкции HN1.

Поскольку hp_key отличается от ключа защиты пакета, защита заголовка обеспечивает уровень AE2, как определено в [NAN], и поэтому гарантирует конфиденциальность field - защищённого заголовка пакета. Будущие варианты защиты заголовка на основе этой конструкции **должны** использовать PRF для обеспечения эквивалентных гарантий защиты.

Использование одного ключа и выборки шифрованных данных более одного раза ставит под угрозу защиту заголовка. Защита двух разных заголовков с одним ключом и выборкой шифротекста раскрывает XOR защищённых полей. В предположении AEAD в качестве PRF и выборки L битов вероятность совпадения двух выборок шифротекста составит $2^{(-L/2)}$ (birthday bound). Для алгоритмов, описанных в этом документе вероятность составляет 2^{-64}.

Чтобы злоумышленник не могу изменить заголовки пакетов, для них применяется транзитивная проверка подлинности с использованием защиты пакета, весь заголовок является частью дополнительных аутентифицируемых данных. Фальсифицированные или изменённые поля, которые были защищены, могут быть обнаружены лишь после снятия защиты пакета.

9.5. Побочные каналы синхронизации защиты заголовка

Атакующий может угадать значения номеров пакетов или Key Phase и получить от конечной точки подтверждение догадки через побочный канал синхронизации. Точно так же можно попытаться угадать и раскрыть номер пакета. Если получатель отбрасывает пакеты с дубликатами номеров, не пытаясь снять защиту пакета, он может раскрыть через побочный канал синхронизации совпадение номера с принятым пакетом. Для устранения побочных каналов при аутентификации процессы снятия защиты заголовка, восстановления порядкового номера и снятия защиты пакета **должны** применяться совместно без синхронизации и других побочных каналов.

Передача пакетов, создание и защита данных и порядковых номеров в пакете **должны** не иметь побочных каналов, которые могут раскрывать номер пакета или его кодированный размер.

В процессе обновления ключей время генерации новых ключей может раскрывать через побочные каналы синхронизации факт обновления ключей. При внедрении пакетов атакующим этот сторонний канал может дополнительно раскрывать значение Key Phase на внедрённых пакетах. После получения обновления ключей конечной точке **следует** создать и сохранить набор ключей защиты приёма, как описано в параграфе 6.3. За счёт генерации новых ключей до получения обновления ключей приём пакетов не будет создавать сигналов синхронизации, через которые возможна утечка Key Phase.

Это зависит от того, генерируются ли ключи в процессе обработки пакета и может потребовать от конечной точки поддержки трёх наборов ключей защиты пакета на приёме для предыдущей, текущей и будущей Key Phase. Конечные точки могут отложить создание следующих ключей защиты пакета на приёме до момента отбрасывания старых ключей, чтобы хранить лишь два набора.

9.6. Разнообразие ключей

При использовании TLS применяется центральное планирование ключей TLS. В результате вовлечения сообщений согласования TLS в расчёт секретов, включение расширения для транспортных параметров QUIC гарантирует, что ключи согласования и 1-RTT не совпадут с ключами, которые может создать сервер, использующий TLS с протоколом TCP. Для предотвращения возможности межпротокольной синхронизации ключей применяются дополнительные меры, улучшающие разделение ключей.

В QUIC ключи защиты пакетов и IV выводятся с использованием иных меток, нежели для аналогичных ключей в TLS. Для сохранения такого разделения новым версиям QUIC **следует** задавать иные метки при создании ключей для защиты пакетов и IV, а также ключей защиты заголовков. В этой версии QUIC применяется строка quic, а другие версии могут применять взамен определяемую версией строку.

Для начальных секретов применяется ключ, зависящий от согласованной версии QUIC. Новым версиям QUIC **следует** задавать иную затравку (salt) для расчёта начальных секретов.

9.7. Случайные значения

QUIC зависит от способности конечных точек создавать безопасные случайные значения напрямую для таких параметров протокола как идентификаторы соединений и опосредованно через TLS. Рекомендации по созданию случайных чисел приведены в [RFC4086].

10. Взаимодействие с IANA

Агентство IANA зарегистрировало код 57 (0x39) для расширения quic_transport_parameters (см. параграф 8.2) в реестре TLS ExtensionType Values [TLS-REGISTRIES]. В столбце Recommended для этого расширения указано значение Yes, а столбец TLS 1.3 включает CH (ClientHello) и EE (EncryptedExtensions).

Таблица 2. Записи реестра TLS ExtensionType Values.

Значение	***Имя расширения***	***TLS 1.3***	***Рекомендация***	***Документ***
57	quic_transport_parameters	CH, EE	YY	Данный документ

11. Литература

11.1. Нормативные документы

[AEAD] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[AES] "Advanced encryption standard (AES)", National Institute of Standards and Technology report, DOI 10.6028/nist.fips.197, November 2001, <https://doi.org/10.6028/nist.fips.197>.

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[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>.

[HKDF] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[QUIC-RECOVERY] 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>.

[QUIC-TRANSPORT] 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>.

[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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[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>.

[SHA] Dang, Q., "Secure Hash Standard", National Institute of Standards and Technology report, DOI 10.6028/nist.fips.180-4, July 2015, <https://doi.org/10.6028/nist.fips.180-4>.

[TLS-REGISTRIES] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

[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>.

11.2. Дополнительная литература

[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>.

[ASCII] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[CCM-ANALYSIS] Jonsson, J., "On the Security of CTR + CBC-MAC", Selected Areas in Cryptography, SAC 2002, Lecture Notes in Computer Science, vol 2595, pp. 76-93, DOI 10.1007/3-540-36492-7_7, 2003, <https://doi.org/10.1007/3-540-36492-7_7>.

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

[GCM-MU] Hoang, V., Tessaro, S., and A. Thiruvengadam, "The Multi-user Security of GCM, Revisited: Tight Bounds for Nonce Randomization", CCS '18: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1429-1440, DOI 10.1145/3243734.3243816, 2018, <https://doi.org/10.1145/3243734.3243816>.

[HTTP-REPLAY] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/info/rfc8470>.

[HTTP2-TLS13] Benjamin, D., "Using TLS 1.3 with HTTP/2", RFC 8740, DOI 10.17487/RFC8740, February 2020, <https://www.rfc-editor.org/info/rfc8740>.

[IMC] Katz, J. and Y. Lindell, "Introduction to Modern Cryptography, Second Edition", ISBN 978-1466570269, 6 November 2014.

[NAN] Bellare, M., Ng, R., and B. Tackmann, "Nonces Are Noticed: AEAD Revisited", Advances in Cryptology - CRYPTO 2019, Lecture Notes in Computer Science, vol 11692, pp. 235-265, DOI 10.1007/978-3-030-26948-7_9, 2019, <https://doi.org/10.1007/978-3-030-26948-7_9>.

[QUIC-HTTP] Bishop, M., Ed., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <https://tools.ietf.org/html/draft-ietf-quic-http-34>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[ROBUST] Fischlin, M., Günther, F., and C. Janson, "Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3", 16 May 2020, <https://eprint.iacr.org/2020/718>.

Приложение A. Пример защиты пакета

В этом приложении даны примеры защиты пакетов для поэтапной проверки реализаций. Определены образцы пакетов Initial для клиента и сервера, а также пакета Retry. В пакетах используется выбранное клиентом значение Destination Connection ID of 0x8394c8f03e515708. Включены также некоторые промежуточные значения. Все числа представлены в шестнадцатеричном формате.

A.1. Ключи

Ниже приведены метка, создаваемые в процессе выполнения функции HKDF-Expand-Label (т. е. HkdfLabel.label) и часть значения, переданного функции HKDF-Expand.

```
client in:  00200f746c73313320636c69656e7420696e00
server in:  00200f746c7331332073657276657220696e00
quic key:   00100e746c733133207175696320656b657900
quic iv:    000c0d746c733133207175696320697600
quic hp:    00100d746c733133207175696320687000
```

Общий исходный секрет

```
initial_secret = HKDF-Extract(initial_salt, cid)
    = 7db5df06e7a69e432496adedb0085192
      3595221596ae2ae9fb8115c1e9ed0a44
```

Секреты для защиты пакетов клиента

```
client_initial_secret
    = HKDF-Expand-Label(initial_secret, "client in", "", 32)
    = c00cf151ca5be075ed0ebfb5c80323c4
      2d6b7db67881289af4008f1f6c357aea

key = HKDF-Expand-Label(client_initial_secret, "quic key", "", 16)
    = 1f369613dd76d5467730efcbe3b1a22d

iv  = HKDF-Expand-Label(client_initial_secret, "quic iv", "", 12)
    = fa044b2f42a3fd3b46fb255c

hp  = HKDF-Expand-Label(client_initial_secret, "quic hp", "", 16)
    = 9f50449e04a0e810283a1e9933adedd2
```

Секреты для защиты пакетов сервера

```
server_initial_secret
    = HKDF-Expand-Label(initial_secret, "server in", "", 32)
    = 3c199828fd139efd216c155ad844cc81
      fb82fa8d7446fa7d78be803acdda951b

key = HKDF-Expand-Label(server_initial_secret, "quic key", "", 16)
    = cf3a5331653c364c88f0f379b6067e37

iv  = HKDF-Expand-Label(server_initial_secret, "quic iv", "", 12)
    = 0ac1493ca1905853b0bba03e

hp  = HKDF-Expand-Label(server_initial_secret, "quic hp", "", 16)
    = c206b8d9b9f0f37644430b490eeaa314
```

A.2. Initial от клиента

В пакете Initial от клиента незащищённая часть данных (payload) содержит показанный ниже кадр CRYPTO и кадры PADDING для увеличения размера содержимого до 1162 байтов.

```
060040f1010000ed0303ebf8fa56f129 39b9584a3896472ec40bb863cfd3e868
04fe3a47f06a2b69484c000004130113 02010000c000000010000e00000b6578
616d706c652e636f6dff01000100000a 00080006001d00170018001000070005
04616c706e0005000501000000000033 00260024001d00209370b2c9caa47fba
baf4559fedba753de171fa71f50f1ce1 5d43e994ec74d748002b000302030400
0d0010000e0403050306030203080408 050806002d00020101001c0002400100
```

```
3900320408ffffffffffffffff050480 00ffff07048000ffff08011001048000
75300901100f088394c8f03e51570806 048000ffff
```

Незащищённый заголовок указывает размер 1182 байта - 4-байтовый номер пакета, 1162 байта кадров и 16-байтовый тег аутентификации. Заголовок включает идентификатор соединения и номера пакета (2)

```
c30000000010883 94c8f03e5157080000449e00000002
```

Из вывода защиты данных делается выборка для защиты заголовка. Поскольку в заголовке используется 4-байтовое кодирование порядкового номера, выбираются первые 16 байтов защищённых данных и применяются к заголовку, как показано ниже.

```
sample = d1b1c98dd7689fb8ec11d242b123dc9b

mask = AES-ECB(hp, sample)[0..4]
     = 437b9aec36

header[0] ^= mask[0] & 0x0f
     = c0
header[18..21] ^= mask[1..4]
     = 7b9aec34
header = c000000001088394c8f03e5157080000449e7b9aec34
```

Полученный в результате защищённый пакет имеет вид

```
c000000001088394c8f03e5157080000 449e7b9aec34d1b1c98dd7689fb8ec11
d242b123dc9bd8bab936b47d92ec356c 0bab7df5976d27cd449f63300099f399
1c260ec4c60d17b31f8429157bb35a12 82a643a8d2262cad67500cadb8e7378c
8eb7539ec4d4905fed1bee1fc8aafba1 7c750e2c7ace01e6005f80fcb7df6212
30c83711b39343fa028cea7f7fb5ff89 eac2308249a02252155e2347b63d58c5
457afd84d05dfffdb20392844ae81215 4682e9cf012f9021a6f0be17ddd0c208
4dce25ff9b06cde535d0f920a2db1bf3 62c23e596d11a4f5a6cf3948838a3aec
4e15daf8500a6ef69ec4e3feb6b1d98e 610ac8b7ec3faf6ad760b7bad1db4ba3
485e8a94dc250ae3fdb41ed15fb6a8e5 eba0fc3dd60bc8e30c5c4287e53805db
059ae0648db2f64264ed5e39be2e20d8 2df566da8dd5998ccabdae053060ae6c
7b4378e846d29f37ed7b4ea9ec5d82e7 961b7f25a9323851f681d582363aa5f8
9937f5a67258bf63ad6f1a0b1d96dbd4 faddfcefc5266ba6611722395c906556
be52afe3f565636ad1b17d508b73d874 3eeb524be22b3dcbc2c7468d54119c74
68449a13d8e3b95811a198f3491de3e7 fe942b330407abf82a4ed7c1b311663a
c69890f4157015853d91e923037c227a 33cdd5ec281ca3f79c44546b9d90ca00
f064c99e3dd97911d39fe9c5d0b23a22 9a234cb36186c4819e8b9c5927726632
291d6a418211cc2962e20fe47feb3edf 330f2c603a9d48c0fcb5699dbfe58964
25c5bac4aee82e57a85aaf4e2513e4f0 5796b07ba2ee47d80506f8d2c25e50fd
14de71e6c418559302f939b0e1abd576 f279c4b2e0feb85c1f28ff18f58891ff
ef132eef2fa09346aee33c28eb130ff2 8f5b766953334113211996d20011a198
e3fc433f9f2541010ae17c1bf202580f 6047472fb36857fe843b19f5984009dd
c324044e847a4f4a0ab34f719595de37 252d6235365e9b84392b061085349d73
203a4a13e96f5432ec0fd4a1ee65accd d5e3904df54c1da510b0ff20dcc0c77f
cb2c0e0eb605cb0504db87632cf3d8b4 dae6e705769d1de354270123cb11450e
fc60ac47683d7b8d0f811365565fd98c 4c8eb936bcab8d069fc33bd801b03ade
a2e1fbc5aa463d08ca19896d2bf59a07 1b851e6c239052172f296bfb5e724047
90a2181014f3b94a4e97d117b4381303 68cc39dbb2d198065ae3986547926cd2
162f40a29f0c3c8745c0f50fba3852e5 66d44575c29d39a03f0cda721984b6f4
40591f355e12d439ff150aab7613499d bd49adabc8676eef023b15b65bfc5ca0
6948109f23f350db82123535eb8a7433 bdabcb909271a6ecbcb58b936a88cd4e
8f2e6ff5800175f113253d8fa9ca8885 c2f552e657dc603f252e1a8e308f76f0
be79e2fb8f5d5fbbe2e30ecadd220723 c8c0aea8078cdfcb3868263ff8f09400
54da48781893a7e49ad5aff4af300cd8 04a6b6279ab3ff3afb64491c85194aab
760d58a606654f9f4400e8b38591356f bf6425aca26dc85244259ff2b19c41b9
f96f3ca9ec1dde434da7d2d392b905dd f3d1f9af93d1af5950bd493f5aa731b4
056df31bd267b6b90a079831aaf579be 0a39013137aac6d404f518cfd4684064
7e78bfe706ca4cf5e9c5453e9f7cfd2b 8b4c8d169a44e55c88d4a9a7f9474241
e221af44860018ab0856972e194cd934
```

A.3. Initial от сервера

Сервер передаёт в ответ показанный ниже пакет, включающий кадры ACK и CRYPTO без заполнения PADDING.

```
0200000000600405a020000560303ee fce7f7b37ba1d1632e96677825ddf739
88cfc79825df566dc5430b9a045a1200 130100002e00330024001d00209d3c94
0d89690b84d08a60993c144eca684d10 81287c834d5311bcf32bb9da1a002b00
020304
```

Заголовок от сервера включает новый идентификатор соединения и 2-байтовый номера пакета (1)

```
c1000000010008f067a5502a4262b50040750001
```

После защиты пакета делается выборка, начиная с третьего байта

```
sample = 2cd0991cd25b0aac406a5816b6394100
mask   = 2ec0d8356a
header = cf000000010008f067a5502a4262b5004075c0d9
```

Защищённый пакет в результате имеет вид

```
cf000000010008f067a5502a4262b500 4075c0d95a482cd0991cd25b0aac406a
5816b6394100f37a1c69797554780bb3 8cc5a99f5ede4cf73c3ec2493a1839b3
dbcba3f6ea46c5b7684df3548e7ddeb9 c3bf9c73cc3f3bded74b562bfb19fb84
022f8ef4cdd93795d77d06edbb7aaf2f 58891850abbdca3d20398c276456cbc4
2158407dd074ee
```

A.4. Пакет Retry

Ниже показан пакет Retry, который может быть передан в ответ на пакет Initial из Приложения A.2. Защита целостности включает выбранный клиентом идентификатор соединения 0x8394c8f03e515708, но это значение не включается в окончательный пакет Retry, показанный ниже.

```
ff000000010008f067a5502a4262b574 6f6b656e04a265ba2eff4d829058fb3f
0f2496ba
```

A.5. Пакет с коротким заголовком ChaCha20-Poly1305

В этом примере показаны некоторые шаги, требуемые для защиты пакета с коротким заголовком. Применяется шифрование AEAD_CHACHA20_POLY1305. В примере TLS создаёт для приложения секрет записи, из которого сервер с помощью HKDF-Expand-Label создаёт 4 значения - ключ, IV, ключ защиты заголовка и секрет, применяемый после обновления ключей (это значение далее в примере не используется).

```
secret
    = 9ac312a7f877468ebe69422748ad00a1
      5443f18203a07d6060f688f30f21632b

key = HKDF-Expand-Label(secret, "quic key", "", 32)
    = c6d98ff3441c3fe1b2182094f69caa2e
      d4b716b65488960a7a984979fb23e1c8

iv  = HKDF-Expand-Label(secret, "quic iv", "", 12)
    = e0459b3474bdd0e44a41c144

hp  = HKDF-Expand-Label(secret, "quic hp", "", 32)
    = 25a282b9e82f06f21f488917a4fc8f1b
      73573685608597d0efcb076b0ab7a7a4

ku  = HKDF-Expand-Label(secret, "quic ku", "", 32)
    = 1223504755036d556342ee9361d25342
      1a826c9ecdf3c7148684b36b714881f9
```

Ниже показаны шаги, требуемые для защиты минимального пакета с пустым полем Destination Connection ID. Этот пакет содержит один кадр PING (содержимое 0x01) и имеет номер 654360564. В примере использование номера пакета размером 3 (кодируется в 49140) позволяет избежать дополнения содержимого пакета, если же использовать более короткое представление номера, потребуются кадры PADDING.

```
pn                 = 654360564 (десятичное)
nonce              = e0459b3474bdd0e46d417eb0
unprotected header = 4200bff4
payload plaintext  = 01
payload ciphertext = 655e5cd55c41f69080575d7999c25a5bfb
```

Полученный шифротекст имеет минимальный размер. При выборке для защиты заголовка первый байт шифротекста пропускается.

```
sample = 5e5cd55c41f69080575d7999c25a5bfb
mask   = aefefe7d03
header = 4cfe4189
```

Защищённый пакет имеет минимальный возможный размер 21 байт.

```
packet = 4cfe4189655e5cd55c41f69080575d7999c25a5bfb
```

Приложение B. Анализ алгоритмов AEAD

В этом приложении описан анализ, используемый при выводе ограничений AEAD для алгоритмов AEAD_AES_128_GCM, AEAD_AES_128_CCM, AEAD_AES_256_GCM. В описаниях используются символы умножения (*), деления (/), возведения в степень (^) и скобки для указания порядка действий. Описания символов приведены ниже.

t
: Размер тега аутентификации с битах. Для рассматриваемых шифров это 128.

n
: Размер блока в битах. Для рассматриваемых шифров это 128.

k
: Размер ключа в битах. Это 128 для AEAD_AES_128_GCM и AEAD_AES_128_CCM, 256 для AEAD_AES_256_GCM.

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 будут принимать конечные точки. Это значение ограничивает число поддельных пакетов, которые конечная точка может отбросить до смены ключей.

o
: Число автономных (offline) запросов шифрования, предпринимаемых злоумышленником.

Последующий анализ основан на подсчете числа операций с блоками при создании каждого сообщения. Этот анализ выполняется для пакетов размером до 2^{11} (l = 2^7) и 2^{16} (l = 2^{12}) байтов. Предполагается, что размер 2^{11} будет пределом для базовых вариантов развёртывания, а 2^{16} задаёт максимальный размер пакета QUIC. Лишь точки, строго ограничивающие размер пакетов, могут использовать более высокие пределы в части целостности и конфиденциальности, выведенные для меньших размеров.

Для AEAD_AES_128_GCM и AEAD_AES_256_GCM размером сообщения (l) считается размер связанных данных в блоках с добавлением размера нешифрованных данных (plaintext) в блоках.

Для AEAD_AES_128_CCM общее число блочных операций шифрования является суммой размера связанных данных в блоках, размера шифротекста в блоках, размера открытых данных в блоках и 1. В этом анализе принимается упрощения до удвоенного размера пакета в блоках (т. е. $2l = 2^8$ для пакетов до 2^{11} байтов и $2l = 2^{13}$ в остальных случаях). Это упрощение основано на пакетах, содержащих все связанные данные и шифротекст, что ведёт к завышению оценки числа операций для пакета на 1-3 блока.

B.1. Анализ применимости AEAD_AES_128_GCM и AEAD_AES_256_GCM

[GCM-MU] определяет конкретные границы применимости AEAD_AES_128_GCM и AEAD_AES_256_GCM при использовании в TLS 1.3 и QUIC. В этом параграфе приведён анализ в некоторыми упрощениями.

- Число шифрованных блоков, используемых атакующим в попытках подделки, ограничено произведением v * l (число попыток подделки, умноженное на размер каждого пакета в блоках).
- Объем работы, автономно выполняемой злоумышленником, не преобладает над другими факторами.

Ограничения в [GCM-MU] являются более жёсткими и полными по сравнению с [AEBounds], где разрешены пределы выше описанных в [TLS13].

B.1.1. Ограничение в части конфиденциальности

В части конфиденциальности теорема 4.3 в [GCM-MU] показывает, что для одного пользователя, не повторяющего значения nonce, доминирующим членом, определяющим сравнительные преимущества реального и случайного алгоритма AEAD, получаемые злоумышленником, составляют

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

Для целевого преимущества 2^{-57} это даёт

```
q <= 2^35 / l
```

Таким образом, конечные точки, не передающие пакетов размером более 2^{11} байтов, не могут защитить более 2^{28} пакетов в одном соединении, не предоставляя атакующему преимущества выше 2^{-57}. Для конечных точек, поддерживающих размер пакетов до 2^{16} байтов, предел составляет 2^{23}.

B.1.2. Ограничение в части целостности

В части целостности теорема 4.3 в [GCM-MU] показывает, что атакующий получит преимущество в подделке пакетов не позднее указанного ниже числа пакетов.

```
(1 / 2^(8 * n)) + ((2 * v) / 2^(2 * n))
        + ((2 * o * v) / 2^(k + n)) + (n * (v + (v * l)) / 2^k)
```

Цель состоит в ограничении этого преимущества значением 2^{-57}. Для AEAD_AES_128_GCM четвёртый член выражения доминирует над остальными, поэтому их можно исключить без существенного влияния на результат. Это даёт приближение

```
v <= 2^64 / l
```

Конечные точки, не пытающиеся снимать защиту с пакетов размером более 2^{11} байтов, могут снимать защиту не более чем с 2^{57} пакетов. Конечные точки, не ограничивающие размер обрабатываемых пакетов, могут снять защиту не более чем с 2^{52} пакетов.

Для AEAD_AES_256_GCM в выражении доминирует тот же член, но большее значение k даёт приближение

```
v <= 2^192 / l
```

Это существенно выше предела для AEAD_AES_128_GCM, однако этот документ рекомендует одинаковый предел для обеих функций, поскольку оба из значений достаточно велики. .

B.2. Анализ применимости AEAD_AES_128_CCM

TLS [TLS13] и [AEBounds] не задают ограничений на использование AEAD_AES_128_CCM. Однако любой алгоритм AEAD, используемый с QUIC, требует ограничений, обеспечивающих сохранение конфиденциальности и целостности. В этом разделе приведён анализ этих ограничений, основанный на [CCM-ANALYSIS]. Результаты этого анализа применяются для вывода пределов применимости в соответствии с ограничениями, заданными в [TLS13].

В части конфиденциальности теорема 2 из [CCM-ANALYSIS] указывает, что атакующий получает заметное преимущество над идеальной псевдослучайной перестановкой (pseudorandom permutation или PRP) после не более чем $(2l * q)^2/2^n$ пакетов.

В части целостности теорема 1 из [CCM-ANALYSIS] указывает, что атакующий получает строго большее преимуществу при том же числе пакетов. Поскольку цели получения преимуществ в части конфиденциальности и целостности совпадают, достаточно рассмотреть лишь теорему 1. Эта теорема показывает, что атакующий получает преимущество перед идеальной PRP при числе пакетов не более

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

Поскольку t и n имеют значение 128, первый член пренебрежимо мал по сравнению со вторым и его можно исключить без существенного влияния на результат. Это порождает отношение, объединяющее попытки шифрования и дешифровки с одинаковым пределом, который определяется лишь теоремой для конфиденциальности. Для целевого преимущества 2^{-57} это даёт

```
v + q <= 2^34,5 / l
```

Установив q = v, можно получить значения для целостности и конфиденциальности. Конечные точки, ограничивающие размер пакетов значением 2^{11} байтов, будут иметь в части целостности и конфиденциальность ограничение $2^{26,5}$ пакетов. Конечные точки, не ограничивающие размер пакетов, будут иметь предел $2^{21,5}$ пакетов.

Участники работы

Рабочая группа IETF QUIC получила поддержку от многих людей. Ниже перечислены те, кто внёс существенных вклад в работу.

Adam Langley
Alessandro Ghedini
Christian Huitema
Christopher Wood
David Schinazi
Dragana Damjanovic
Eric Rescorla
Felix Günther
Ian Swett
Jana Iyengar
奥 一穂 (Kazuho Oku)
Marten Seemann
Martin Duke
Mike Bishop
Mikkel Fahnøe Jørgensen
Nick Banks
Nick Harper
Roberto Peon
Rui Paulo
Ryan Hamilton
Victor Vasiliev

Адреса авторов

Martin Thomson (editor)

Mozilla

Email: mt@lowentropy.net

Sean Turner (editor)

sn3rd

Email: sean@sn3rd.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru