Abstract

This document specifies QUIC version 2, which is identical to QUIC version 1 except for some trivial details. Its purpose is to combat various ossification vectors and exercise the version negotiation framework. It also serves as a template for the minimum changes in any future version of QUIC.

Note that "version 2" is an informal name for this proposal that indicates it is the second standards-track QUIC version. The protocol specified here uses a version number other than 2 in the wire image.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://quicwg.org/quic-v2/draft-ietf-quic-v2.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-quic-v2/.

Discussion of this document takes place on the QUIC Working Group mailing list (mailto:quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/.

Source for this draft and an issue tracker can be found at https://github.com/quicwg/quic-v2.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.
Table of Contents

1. Introduction ........................................... 3
2. Conventions ............................................ 3
3. Differences with QUIC Version 1 ........................... 4
   3.1. Version Field ..................................... 4
   3.2. Long Header Packet Types ......................... 4
   3.3. Cryptography changes ............................... 4
      3.3.1. Initial Salt .................................. 4
      3.3.2. HKDF Labels .................................. 4
      3.3.3. Retry Integrity Tag .......................... 5
4. Version Negotiation Considerations ........................ 5
   4.1. Compatible Negotiation Requirements ............... 5
5. TLS Resumption and NEW_TOKEN Tokens ....................... 6
6. Ossification Considerations ................................ 7
7. Applicability ........................................... 7
8. Security Considerations .................................. 7
9. IANA Considerations ..................................... 7
10. References ............................................... 8
    10.1. Normative References ............................ 8
    10.2. Informative References .......................... 8
Appendix A. Sample Packet Protection ........................ 9
1. Introduction

QUIC [QUIC] has numerous extension points, including the version number that occupies the second through fifth bytes of every long header (see [QUIC-INVARIANTS]). If experimental versions are rare, and QUIC version 1 constitutes the vast majority of QUIC traffic, there is the potential for middleboxes to ossify on the version bytes always being 0x00000001.

In QUIC version 1, Initial packets are encrypted with the version-specific salt as described in Section 5.2 of [QUIC-TLS]. Protecting Initial packets in this way allows observers to inspect their contents, which includes the TLS Client Hello or Server Hello messages. Again, there is the potential for middleboxes to ossify on the version 1 key derivation and packet formats.

Finally [QUIC-VN] provides two mechanisms for endpoints to negotiate the QUIC version to use. The "incompatible" version negotiation method can support switching from any QUIC version to any other version with full generality, at the cost of an additional round-trip at the start of the connection. "Compatible" version negotiation eliminates the round-trip penalty but levies some restrictions on how much the two versions can differ semantically.

QUIC version 2 is meant to mitigate ossification concerns and
exercise the version negotiation mechanisms. The only change is a
tweak to key derivation inputs to enforce full key separation. Any
endpoint that supports two versions needs to implement version
negotiation to protect against downgrade attacks.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
"OPTIONAL" in this document are to be interpreted as described in
BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all
capitals, as shown here.

3. Differences with QUIC Version 1

QUIC version 2 endpoints MUST implement the QUIC version 1
specification as described in [QUIC], [QUIC-TLS], and
[QUIC-LOSSRECOVERY]. However, the following differences apply in
version 2.

3.1. Version Field

The Version field of long headers is 0x709a50c4.

*RFC Editor's Note:* Please remove the sentence below prior to
publication of a final version of this document.

This version number will not change in subsequent versions of this
draft, unless there is a behavior change that breaks compatibility.

3.2. Long Header Packet Types

All version 2 long header packet types are different. The Type field
values are:

* Initial: 0b01
* 0-RTT: 0b10
* Handshake: 0b11
* Retry: 0b00
3.3. Cryptography changes

3.3.1. Initial Salt

The salt used to derive Initial keys in Section 5.2 of [QUIC-TLS] changes to:

\[ \text{initial\_salt} = 0xa707c203a59b47184a1d62ca570406ea7ae3e5d3 \]

3.3.2. HKDF Labels

The labels used in [QUIC-TLS] to derive packet protection keys (Section 5.1), header protection keys (Section 5.4), Retry Integrity Tag keys (Section 5.8), and key updates (Section 6.1) change from "quic key" to "quicv2 key", from "quic iv" to "quicv2 iv", from "quic hp" to "quicv2 hp", and from "quic ku" to "quicv2 ku", to meet the guidance for new versions in Section 9.6 of that document.

3.3.3. Retry Integrity Tag

The key and nonce used for the Retry Integrity Tag (Section 5.8 of [QUIC-TLS]) change to:

\[ \text{secret} = 0x3425c20cf88779df2ff71e8abfa78249891e763bbed2f13c048343d348c060e2 \]
\[ \text{key} = 0xba858dc7b43de5dbf87617ff4ab253db \]
\[ \text{nonce} = 0x141b99c239b03e785d6a2e9f \]

4. Version Negotiation Considerations

QUIC version 2 is not intended to deprecate version 1. Endpoints that support version 2 might continue support for version 1 to maximize compatibility with other endpoints. In particular, HTTP clients often use Alt-Svc [RFC7838] to discover QUIC support. As this mechanism does not currently distinguish between QUIC versions, HTTP servers that support multiple versions reduce the probability of incompatibility and the cost associated with QUIC version negotiation or TCP fallback. For example, an origin advertising support for "h3" in Alt-Svc SHOULD support QUIC version 1 as it was the original QUIC version used by HTTP/3 and therefore some clients will only support
that version.

Any QUIC endpoint that supports QUIC version 2 MUST send, process, and validate the version_information transport parameter specified in [QUIC-VN] to prevent version downgrade attacks.

Note that version 2 meets that document's definition of a compatible version with version 1, and version 1 is compatible with version 2. Therefore, servers can use compatible negotiation to switch a connection between the two versions. Endpoints that support both versions SHOULD support compatible version negotiation to avoid a round trip.

4.1. Compatible Negotiation Requirements

Compatible version negotiation between versions 1 and 2 follow the same requirements in either direction. This section uses the terms "original version" and "negotiated version" from [QUIC-VN].

If the server sends a Retry packet, it MUST use the original version. The client ignores Retry packets using other versions. The client MUST NOT use a different version in the subsequent Initial that contains the Retry token. The server MAY encode the QUIC version in its Retry token to validate that the client did not switch versions, and drop the packet if it switched.

QUIC version 2 uses the same transport parameters to authenticate the Retry as QUIC version 1. After switching to a negotiated version after a Retry, the server MUST include the relevant transport parameters to validate that the server sent the Retry and the connection IDs used in the exchange, as described in Section 7.3 of [QUIC].

The server SHOULD start sending its Initial packets using the negotiated version as soon as it decides to change. Before the server is able to process transport parameters from the client, it might need to respond to Initial packets from the client. For these packets the server uses the original version.

Once the client has processed a packet using the negotiated version, it SHOULD send subsequent Initial packets using that version. The
server MUST NOT discard its original version Initial receive keys until it successfully processes a packet with the negotiated version.

Both endpoints MUST send Handshake or 1-RTT packets using the negotiated version. An endpoint MUST drop packets using any other version. Endpoints have no need to generate the keying material that would allow them to decrypt or authenticate these packets.

The client MUST NOT send 0-RTT packets using the negotiated version, even after processing a packet of that version from the server. Servers can apply original version 0-RTT packets to a connection without additional considerations.

5. TLS Resumption and NEW_TOKEN Tokens

TLS session tickets and NEW_TOKEN tokens are specific to the QUIC version of the connection that provided them. Clients SHOULD NOT use a session ticket or token from a QUIC version 1 connection to initiate a QUIC version 2 connection, or vice versa.

Servers MUST validate the originating version of any session ticket or token and not accept one issued from a different version. A rejected ticket results in falling back to a full TLS handshake, without 0-RTT. A rejected token results in the client address remaining unverified, which limits the amount of data the server can send.

After compatible version negotiation, any resulting session ticket maps to the negotiated version rather than original one.

6. Ossification Considerations

QUIC version 2 provides protection against some forms of ossification. Devices that assume that all long headers will encode version 1, or that the version 1 Initial key derivation formula will remain version-invariant, will not correctly process version 2 packets.
However, many middleboxes such as firewalls focus on the first packet in a connection, which will often remain in the version 1 format due to the considerations above.

Clients interested in combating firewall ossification can initiate a connection using version 2 if they are either reasonably certain the server supports it, or are willing to suffer a round-trip penalty if they are incorrect. In particular, a server that issues a session ticket for version 2 indicates an intent to maintain version 2 support while the ticket remains valid, even if support cannot be guaranteed.

7. Applicability

This version of QUIC provides no change from QUIC version 1 relating to the capabilities available to applications. Therefore, all Application Layer Protocol Negotiation (ALPN) ([RFC7301]) codepoints specified to operate over QUIC version 1 can also operate over this version of QUIC. In particular, both the "h3" [I-D.ietf-quic-http] and "doq" [RFC9250] ALPNs can operate over QUIC version 2.

Unless otherwise stated, all QUIC extensions defined to work with version 1 also work with version 2.

8. Security Considerations

QUIC version 2 introduces no changes to the security or privacy properties of QUIC version 1.

The mandatory version negotiation mechanism guards against downgrade attacks, but downgrades have no security implications, as the version properties are identical.

9. IANA Considerations

This document requests that IANA add the following entry to the QUIC version registry:

Value: 0x709a50c4
10. References

10.1. Normative References


10.2. Informative References
Appendix A. Sample Packet Protection

This section shows examples of packet protection so that implementations can be verified incrementally. Samples of Initial packets from both client and server plus a Retry packet are defined. These packets use an 8-byte client-chosen Destination Connection ID of 0x8394c8f03e515708. Some intermediate values are included. All values are shown in hexadecimal.

A.1. Keys

The labels generated during the execution of the HKDF-Expand-Label function (that is, HkdfLabel.label) and part of the value given to the HKDF-Expand function in order to produce its output are:

client in: 00200f746c73313320636c69656e7420696e00
server in: 00200f746c7331332073657276657220696e00
quicv2 key: 001010746c73313320717569637632206b657900
quicv2 iv: 000c0f746c7331332071756963763220697600

The initial secret is common:

\[
\text{initial\_secret} = \text{HKDF-Extract}(\text{initial\_salt}, \text{cid})
\]
\[
= \text{ddfcb7b82a430b7845210ad64b406977}
\]
\[
= \text{ed51b269a14bc69aa9ea9b366fa3b06b}
\]

The secrets for protecting client packets are:

\[
\text{client\_initial\_secret} = \text{HKDF-Expand-Label}(\text{initial\_secret}, "client in", ",", 32)
\]
\[
= \text{9fe72e1452e91f551b770005054034e4}
\]
\[
= \text{7575d4a0fb4c27b7c6cb303a338423ae}
\]

\[
\text{key} = \text{HKDF-Expand-Label}(\text{client\_initial\_secret}, "quicv2 key", ",", 16)
\]
\[
= \text{95df2be2e8d549c82e996fc9339f4563}
\]

\[
\text{iv} = \text{HKDF-Expand-Label}(\text{client\_initial\_secret}, "quicv2 iv", ",", 12)
\]
\[
= \text{ea5e3c95f933db14b7020ad8}
\]

\[
\text{hp} = \text{HKDF-Expand-Label}(\text{client\_initial\_secret}, "quicv2 hp", ",", 16)
\]
\[
= \text{091efb735702447d07908f6501845794}
\]

The secrets for protecting server packets are:

\[
\text{server\_initial\_secret} = \text{HKDF-Expand-Label}(\text{initial\_secret}, "server in", ",", 32)
\]
\[
= \text{3c9bf6a9c1c8c71819876967bd8b979e}
\]
\[
= \text{fd98ec665edf27f22c06e9845ba0ae2f}
\]

\[
\text{key} = \text{HKDF-Expand-Label}(\text{server\_initial\_secret}, "quicv2 key", ",", 16)
\]
\[
= \text{15d5b4d9a2b8916aa39b1bfe574d2aad}
\]

\[
\text{iv} = \text{HKDF-Expand-Label}(\text{server\_initial\_secret}, "quicv2 iv", ",", 12)
\]
\[
= \text{a85e7ac31cd275cbb095c626}
\]

\[
\text{hp} = \text{HKDF-Expand-Label}(\text{server\_initial\_secret}, "quicv2 hp", ",", 16)
\]
\[
= \text{b13861cfadbb9d11ff942dd80c8fc33b}
\]
A.2. Client Initial

The client sends an Initial packet. The unprotected payload of this packet contains the following CRYPTO frame, plus enough PADDING frames to make a 1162-byte payload:

```
060040f1010000ed0303ebf8fa56f129 39b9584a3896472ec40bb863cfd3e868
04fe3a47f06a2b69484c000004130113 02010000c000000100000e0000006578
616d706c652e636f6dff010001000000a 00080006001d00170018001000070005
04616c706e00050005000500000033 00260024001d00209370b2c9aa47fba
baf4559fedba753de171fa71f50f1ce1 5d43e994ec74d748002b000302030400
0d000000e0403050306030203000408 050806002d00020101001c0002400100
3900320408ffffffffffffffff050480 0ff0ff07048000fffffff08011001048000
7530090110f088394c8f03e51570806 048000ffff
```

The unprotected header indicates a length of 1182 bytes: the 4-byte packet number, 1162 bytes of frames, and the 16-byte authentication tag. The header includes the connection ID and a packet number of 2:

```
d3709a50c4088394c8f03e5157080000449e0000002
```

Protecting the payload produces output that is sampled for header protection. Because the header uses a 4-byte packet number encoding, the first 16 bytes of the protected payload is sampled and then applied to the header as follows:

```
sample = 23b8e61058c83c92d0e97eb7a6e5003

mask = AES-ECB(hp, sample)[0..4]
    = 8e4391d84a

header[0] ^= mask[0] & 0x0f
    = dd
header[18..21] ^= mask[1..4]
    = 4391d848
header = dd709a50c4088394c8f03e5157080000449e4391d848
```

The resulting protected packet is:
A.3. Server Initial

The server sends the following payload in response, including an ACK frame, a CRYPTO frame, and no PADDING frames:

```
02000000000600405a020000560303ee fce7f7b37ba1d632e96677825ddf739
88cfc79825df566dc54309ba0a45a1200 130100002e00330024001d00209d3c94
020304
```

Duke                     Expires 3 December 2022               [Page 12]

The header from the server includes a new connection ID and a 2-byte packet number encoding for a packet number of 1:

d1709a50c40008f067a5502a4262b50040750001

As a result, after protection, the header protection sample is taken starting from the third protected byte:

```
sample = ebb7972fdce59d50e7e49ff2a7e8de76
mask   = 41103f438e
header = d0709a50c40008f067a5502a4262b5004075103e
```

The final protected packet is then:

d0709a50c40008f067a5502a4262b500 4075103e63b4ebb7972fdce59d50e7e4
9ff2a7e8de76b0cd8c1010a1f3d549 dd6fe801588fb14d279bef8d7c53ef62
66a9a7a1a5f2fa026c236a5bf8df5aa0 f9d74773acecffe9100b0f76814b5e33
f7b7f8ec278d23f8c7a9e66856b8be 72558135bca27c54d63ffcc902253461c
A.4. Retry

This shows a Retry packet that might be sent in response to the Initial packet in Appendix A.2. The integrity check includes the client-chosen connection ID value of 0x8394c8f03e515708, but that value is not included in the final Retry packet:

cf709a50c40008f067a5502a4262b574 6f6b656e1dc71130cd1ed39d6efcee5c 85806501

A.5. ChaCha20-Poly1305 Short Header Packet

This example shows some of the steps required to protect a packet with a short header. This example uses AEAD_CHACHA20_POLY1305.

In this example, TLS produces an application write secret from which a server uses HKDF-Expand-Label to produce four values: a key, an IV, a header protection key, and the secret that will be used after keys are updated (this last value is not used further in this example).

secret = 9ac312a7f877468ebe69422748ad00a1
        5443f18203a07d6060f688f30f21632b
key = HKDF-Expand-Label(secret, "quicv2 key", ",", 32)
     = 3bfcedd72bcf02541d7fa0dd1f5f9e32
     a817e09a6963a0e6c7df0f9a1bab90f2
iv = HKDF-Expand-Label(secret, "quicv2 iv", ",", 12)
    = a6b5bc6ab7da5ce30ffffff5dd
hp = HKDF-Expand-Label(secret, "quicv2 hp", ",", 32)
     = d659760d2ba434a226fd37b35c69e2da
             8211d10c4f12538787d65645d5d1b8e2

ku = HKDF-Expand-Label(secret, "quicv2 ku", ",", 32)
     = c69374c49e3d2a9466fa689e49d476db
             5d0dfbc87d32ceeeaa6343fd0ae4c7d88

The following shows the steps involved in protecting a minimal packet
with an empty Destination Connection ID. This packet contains a
single PING frame (that is, a payload of just 0x01) and has a packet
number of 654360564. In this example, using a packet number of
length 3 (that is, 49140 is encoded) avoids having to pad the payload
of the packet; PADDING frames would be needed if the packet number is
encoded on fewer bytes.

pn                 = 654360564 (decimal)
nonce              = a6b5bc6ab7dafce328ff4a29
unprotected header = 4200bff4
payload plaintext  = 01
payload ciphertext = 0ae7b6b932bc27d786f4bc2bb20f2162ba

The resulting ciphertext is the minimum size possible. One byte is
skipped to produce the sample for header protection.

sample = e7b6b932bc27d786f4bc2bb20f2162ba
mask   = 97580e32bf
header = 5558b1c6

The protected packet is the smallest possible packet size of 21
bytes.

packet = 5558b1c60ae7b6b932bc27d786f4bc2bb20f2162ba

Appendix B. Acknowledgments

The author would like to thank Lucas Pardue, David Schinazi, and
Martin Thomson for their helpful suggestions.
Appendix C. Changelog

*RFC Editor's Note:* Please remove this section prior to publication of a final version of this document.

C.1. since draft-ietf-quic-v2-03

* a v2 session ticket is an indication of v2 support
* a v1-only extension is theoretically possible
* editorial fixes

C.2. since draft-ietf-quic-v2-02

* Editorial comments from Lucas Pardue

C.3. since draft-ietf-quic-v2-01

* Ban use of NEW_TOKEN tokens across versions
* version-info transport parameter required for all v2 endpoints
* Explicitly list known ALPN compatibility

C.4. since draft-ietf-quic-v2-00

* Expanded requirements for compatible version negotiation
* Added test vectors
* Greased the packet type codepoints
* Random version number
* Clarified requirement to use QUIC-VN
* Banned use of resumption tokens across versions

C.5. since draft-duke-quic-v2-02

* Converted to adopted draft
* Deleted references to QUIC improvements
* Clarified status of QUIC extensions

C.6. since draft-duke-quic-v2-01
* Made the final version number TBD.
* Added ALPN considerations

C.7. since draft-duke-quic-v2-00
* Added provisional versions for interop
* Change the v1 Retry Tag secret
* Change labels to create full key separation

Author's Address

Martin Duke
Google LLC
Email: martin.h.duke@gmail.com