

quic
Internet-Draft
Intended status: Standards Track
Expires: 29 October 2022

M. Duke
D. Schinazi
Google LLC
27 April 2022

Protected QUIC Initial Packets
draft-duke-quic-protected-initial-04

Abstract

QUIC encrypts its Initial Packets using keys derived from well-known constants, meaning that observers can inspect the contents of these packets and successfully spoof them. This document proposes a new version of QUIC that encrypts Initial Packets more securely by leveraging a Public Key distributed via the Domain Name System (DNS) or other out-of-band system.

Discussion Venues

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

Discussion of this document takes place on the mailing list (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/martinduke/quic-version-aliasing>.

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/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 29 October 2022.

Internet-Draft

protected-initial

April 2022

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.	Introduction	3
1.1.	Relationship to ECH and Version Aliasing	4
2.	Conventions	5
3.	Differences with QUIC Version 1	5
3.1.	Version Number	5
3.2.	Key Configuration	5
3.3.	Key Derivation Labels	5
3.4.	Initial Packet Header	5
3.4.1.	Encryption Context	6
3.5.	Client Packet Protection Procedure	7
3.6.	Server Packet Protection Procedure	7
3.7.	Retry Integrity Tag	8
3.8.	Fallback	8
3.9.	The Fallback Packet	10
3.10.	New Transport Parameters	10
3.10.1.	public_key_failed	10
3.10.2.	ECHConfig	11
3.10.3.	initial_encryption_context	11
4.	Intermediaries	12
4.1.	Client-Facing Server	12
4.2.	Back-End Server	12
5.	Applicability	13
6.	Security and Privacy Considerations	13
6.1.	Forcing Downgrade	13
6.2.	Initial Packet Injection	14
6.3.	Retry Injection	15
7.	IANA Considerations	15

7.1.	QUIC Version Registry	15
7.2.	QUIC Transport Parameter Registry	16
7.3.	QUIC Transport Error Codes Registry	16
8.	References	16
8.1.	Normative References	16

8.2.	Informative References	17
Appendix A.	Change Log	17
A.1.	since draft-duke-quic-protected-initials-01	18
A.2.	since draft-duke-quic-protected-initials-00	18
	Authors' Addresses	18

[1.](#) Introduction

DISCLAIMER: This draft is a preliminary proposal with insufficient security analysis. It should not be used in production systems.

The QUIC Initial Packet [[RFC9000](#)] is encrypted using a key derived from the Destination Connection ID in the packet cleartext and a well-known salt (see [Section 5.2 of \[RFC9001\]](#)). [Section 7](#) of that draft describes security vulnerabilities resulting from the resulting lack of authentication.

This also has privacy implications, as observers can decrypt the packet and inspect the contents, which contain the TLS Client Hello and Server Hello Messages ([\[RFC8446\]](#)). The former contains QUIC Transport Parameters, which reveal even more information about the traffic.

Furthermore, packets vulnerable to deep inspection create an ossification vector. Intermediaries that expect the contents of these messages to match a certain format and template might drop packets that do not, preventing the use of new protocol extensions or improved security protocols.

This document proposes a new version of QUIC where the client obtains a public key generated by the server and uses it to encrypt a shared secret, sent in the Initial packet header, that both endpoints can then use to protect Initial packets.

This mechanism leverages the public key that would be distributed via DNS (or other out-of-band mechanism) to support Encrypted Client

Hello [[ECHO](#)]. That design uses Hybrid Public Key Exchange (HPKE) [[HPKE](#)] to authenticate some HPKE configuration information and the "outer client hello" that is in plaintext, while encrypting the "inner client hello" that contains privacy-sensitive information. This document uses the widespread configuration that will exist if ECHO is widely deployed, but only sends the subset of information necessary to seed the QUIC key generation process.

[1.1](#). Relationship to ECH and Version Aliasing

Encrypted Client Hello [[ECHO](#)] and QUIC Version Aliasing [[VERSION-ALIASING](#)] also exist in the solution space of concealing parts of the Initial packet exchange from observers. The following table summarizes the advantages and disadvantages of each.

Property	ECH	Protected Initials	Version Aliasing
Fields Protected	Some of Client Hello	All Initial Payloads	All Initial Payloads
Delay when server loses its keys	1 RTT	2 RTT	2 RTT
Works with TLS over TCP	Yes	No	No
First-connection protection	Yes	Yes	No
Prevents Initial packet injection attacks	No	Yes	Yes
Symmetric Encryption Only	No	No	Yes

Greases the Version Field	No	No	Yes
Prevents Retry injection attacks	No	No	Yes
No trial decryption	No	No	Yes

Table 1

The more complex properties in the table are discussed in [Section 6](#).

Both ECH and Protected Initials are complimentary with Version Aliasing: they provide a computationally expensive way to protect parts of the Initial packet during the first connection between client and server, after which Version Aliasing can protect future exchanges with several additional desirable properties.

[2.](#) Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

[3.](#) Differences with QUIC Version 1

The version of QUIC described in this specification is identical to QUIC version 1 [[RFC9000](#)] except where described in this document.

[3.1.](#) Version Number

The version field in long headers is TBD. Note: for interoperability exercises, use the provisional value 0xff454900.

[3.2.](#) Key Configuration

The client obtains the Encrypted ClientHello Configuration (ECHConfig) described in Section 4 of [[ECHO](#)], which provides the context that allows protection of Initial packets. The ECHConfig is available via a DNS record or other out-of-band system.

[3.3.](#) Key Derivation Labels

The labels used to derive keying material in [[RFC9001](#)] change from "quic key", "quic iv", "quic hp", and "quic ku" to "quicpi key", "quicpi iv", "quicpi hp", and "quicpi ku", respectively.

[3.4.](#) Initial Packet Header

The figure below is presented in the format from [[RFC9000](#)], and adds a variable-length Encryption Context preceded by a length field:

```
Initial Packet {
  Header From (1) = 1,
  Fixed Big (1) = 1,
  Long Packet Type (2) = 0,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Token Length (i),
  Token (..),
  Encryption Context Length (8),
  Encryption Context (..),
```

```
    Length (i),
    Packet Number (8..32),
}
```

Encryption Context Length: A variable-length integer specifying the length of the encryption context, in bytes. Servers MUST set this field to zero; a client that receives a non-zero length MUST either discard the packet or generate a connection error of type `PROTOCOL_VIOLATION`.

Clients MUST include a nonzero Encryption Context Length in all Initial packets, unless executing fallback procedures (see [Section 3.8](#)).

When the client includes a non-zero-length Encryption Context, it MUST include an `initial_encryption_context` in its Client Hello, so that this header field is included in the TLS handshake transcript.

[3.4.1](#). Encryption Context

The encryption context, if nonzero length, has the following format:

```
Encryption Context {
    Config ID (8),
    SCKDF (16),
    SCAEADID (16),
    Encapsulated Secret (..),
}
```

Config ID: An 8-bit integer identifying the configuration parameters, obtained from the `ECHConfig`. This ID implicitly identifies the public key, Key Encapsulation Mechanism, and the set of symmetric suites from which the following fields are selected.

SCKDF: Symmetric Cipher Key Derivation Function. The client selects this from the cipher suites listed in the `ECHConfig`. This is the cipher used to encrypt the Initial Packet. The values are listed in Section 7.2 of [\[HPKE\]](#). All endpoints MUST support HKDF-SHA256 (0x0001), which is used for QUIC Version 1 Initial packets.

SCAEADID: Symmetric Cipher Key Derivation Function. The client selects this from the cipher suites listed in the `ECHConfig`. This is

the cipher used to encrypt the Initial Packet. The values are listed in Section 7.3 of [\[HPKE\]](#). All endpoints MUST support AES-128-GCM (0x0001), which is used for QUIC Version 1 Initial packets.

The Encapsulated Secret is HPKE encapsulated. Its length is inferred from the Encryption Context Length field.

[3.5.](#) Client Packet Protection Procedure

An client extracts the public key pkR and uses it to generate a shared_secret:

```
pkR = Deserialize(ECHConfig.contents.public_key)
shared_secret, enc = Encap(pkR)
initial_secret = HKDF-Extract(shared_secret,
                               client_dst_connection_id || ECHConfig)
```

enc is the Encapsulated Secret, and is written into that subfield of the Encryption Context Field.

The initial_secret above is used to generate client_initial_secret and server_initial_secret as described in [Section 5.2 of \[RFC9001\]](#).

When applying header protection, the Context Length and Encryption Context are not Protected.

This derivation is performed once per connection. Subsequent Initial Packets use the same keys and the same offset to the packet number, regardless of additional Encryption Context fields or changed connection IDs.

[3.6.](#) Server Packet Protection Procedure

The server reads the Config ID and Encapsulated Secret (enc) from the Initial Packet. It looks up its private key (skR) associated with the Config ID.

Otherwise, the server generates the Initial secrets:

```
shared_secret = Decap(enc, skR)
```



```
initial_secret = HKDF-Extract(shared_secret,  
                               client_dst_connection_id || ECHConfig)
```

The remainder is identical to the client procedure. All server-generated Initial packets use the keys generated in this process.

If packet decryption fails, see [Section 3.8](#).

The server terminates the connection with a `TRANSPORT_PARAMETER_ERROR` under the following conditions:

- * There is a zero-length Encryption Context field, but the `initial_encryption_context` transport parameter is present
- * There is a non-zero-length Encryption Context field, but the `initial_encryption_context` transport parameter is absent or does not match the header.

However, see [Section 4](#) for exceptions to this transport parameter processing rule.

[3.7](#). Retry Integrity Tag

The Retry packet is identical to QUIC version 1, except that the key and nonce used for the Retry Integrity Tag (Sec 5.8 of [\[RFC9001\]](#)) change to:

```
secret = 0xe453a2e22377289f08a4458ee1c9a90a4e39696e026372ffc33190b8de5a0123  
key = 0xbe0c690b9f66575a1d766b54e368c84e  
nonce = 0x461599d35d632bf2239825bb
```

This key and nonce are also used in Fallback packets ([Section 3.9](#)).

[3.8](#). Fallback

If decryption fails, the client may not have the server's correct configuration. In this case, the server replies with a Fallback packet (see [Section 3.9](#)), and discards the Initial.

The client checks the Integrity Tag in the packet against the received Fallback Packet and its own record of the Initial Packet. If they do not match, the packet may have been corrupted, and the client SHOULD treat the packet as lost. If they do match, the client MUST assume that it does not have the correct ECHConfig.

When it has the incorrect config, the client composes a new Client Hello. It MUST include the `public_key_failed` transport parameter with the Config ID and public key it attempted to use. It MUST use an Encryption Context Length of zero, and encrypt it with keys derived from the "fallback salt"

0xbd62319ad6eeb17a9ed0d3bf75e37e4a8e7e6ac7, instead of the shared secret that the server cannot decode. The Client Hello also MUST include any Retry Token the previous Initial contained and MAY include a token from a NEW_TOKEN frame.

This new Initial packet is part of the same connection and MUST use the same Connection IDs and packet number space.

Note that the fallback salt does not provide confidentiality to the client, and the client SHOULD NOT include privacy-sensitive information there. See [Section 6.1](#) for further discussion of this.

A server interprets a client Initial with a zero-length Encryption Context as an Initial encrypted with keys derived from the fallback salt and decrypts it accordingly.

The server checks the Config ID and public key from the `public_key_failed` transport parameter to see if it should successfully process a Protected Initial with these parameters. If it would have, it MUST terminate the connection with an INVALID_PROTECTED_INITIAL_DOWNGRADE error to indicate that there has been an attack.

If the server would not have successfully decoded the packet with those parameters, it MUST send its own `public_key_failed` transport parameter to acknowledge the parameter was successfully processed. It MAY also send a ECHConfig transport parameter to allow use of Protected Initials in subsequent connections, a Version Aliasing transport parameter (see [\[VERSION-ALIASING\]](#)) to enable a different means of Initial Protection, both, or neither.

If the client does not receive a `public_key_failed` transport parameter in response to sending a `public_key_failed` transport parameter, it MUST terminate the connection with a TRANSPORT_PARAMETER_ERROR.

The client MUST NOT include the original client hello in its TLS transcript hash for key computation, as the server has no access to this message. However, the client hello is an input to the Integrity tag in the `public_key_failed` transport parameter, which will be in the transcript.

Internet-Draft

protected-initial

April 2022

If any part of the client hello changes (e.g the SNI or ALPN), the client MUST NOT include a resumption ticket or send 0-RTT packets.

[3.9.](#) The Fallback Packet

The Fallback packet uses the 0x1 packet type code, which it shares with 0-RTT. Since 0-RTT is only sent by clients and Fallback is only sent by servers, these two types are distinguished by the endpoint's role in the handshake.

The Fallback packet has the following format:

```
Fallback Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 1,
  Unused (4),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Integrity Tag (128),
}
```

The server computes the Integrity Tag by prepending the entire UDP payload that contained the client Initial to the Fallback packet contents (minus the tag itself, of course) to form a pseudo-packet. The server then computes the Integrity Tag as the output of AEAD_AES_128_GCM with the key and nonce from [Section 3.7](#), no plaintext, and the pseudo-packet as the Additional Data. This thus confirms the integrity of both packets in the exchange.

[3.10.](#) New Transport Parameters

[3.10.1.](#) public_key_failed

The public_key_failed transport parameter allows detection of an attacker injecting messages to force use of the fallback salt. For

details on use, see [Section 3.8](#).

Its provisional type is 0x706b66.

When sent by a client, the content of the transport parameter is as follows:

```
{
  Integrity Tag (128),
  Config ID (8),
  Public Key (..),
}
```

The Integrity Tag is copied from the Fallback packet.

The other fields are populated using the ECHConfig that the client attempted to use. The length of the Public Key is inferred from the length field of the transport parameter.

When sent by a server, the transport parameter has no value field.

Endpoints MUST respond to a malformed transport parameter by closing the connection with a TRANSPORT_PARAMETER_ERROR.

Note that this transport parameter is always sent as a result of a fallback from a Protected Initial, and therefore with a zero-length Encryption Context in the packet header. Therefore, if received with non-zero-length Encryption Context, the receiving endpoint MUST terminate the connection with a TRANSPORT_PARAMETER_ERROR.

[3.10.2](#). ECHConfig

The ECHConfig transport parameter allows servers to directly provide clients a valid configuration.

Its provisional type is 0x454348.

Its format is equivalent to ECHConfigList, as defined in Section 4 of [\[ECHO\]](#).

If this transport parameter does not match this format, is received by a server, or is received in a connection where the client's most recent Initial had a non-zero-length Encryption Context, the receiver MUST terminate the connection with a `TRANSPORT_PARAMETER_ERROR`.

[3.10.3.](#) `initial_encryption_context`

The format of this transport parameter is identical to the Encryption Context described in [Section 3.4.1](#).

Its provisional type is `0x696563`.

[4.](#) Intermediaries

In the topology proposed in Section 3.1 of [\[ECHO\]](#), where a client-facing server has its own public name and potentially fronts a number of independent domains, only the client-facing server has the private keys. Thus, it modifies incoming Initial Packets before forwarding to the back-end server.

A common use case of this topology is a load balancer fronting multiple domains using the same IP address, which makes routing decisions based on the SNI in the Client Hello.

[4.1.](#) Client-Facing Server

The client-facing server is responsible for verifying the Initial packet is decryptable, sending a Fallback packet (and dropping the Initial) if it is not, and otherwise forwarding the packet encrypted with a different key.

If an incoming Initial packet is not decryptable, the client-facing server sends a Fallback packet and drops the Initial.

If an incoming client Initial has a non-zero length Encryption Context field, the client-facing server MUST delete the Encryption Context field. When forwarding to the back-end server, it encrypts

the plaintext version of this modified packet with keys derived from the fallback salt. Thus, between client-facing server and back-end server the packet is inspectable by observers.

Similarly, if the client is using the shared secret, the client-facing server MUST decrypt server Initial packets encrypted with keys derived from the fallback secret, and re-encrypt them with keys derived from the shared secret.

The client-facing server MUST enforce correct use of the `initial_encryption_context` parameter, as described in [Section 3.6](#), and terminate the connection if necessary.

Non-Initial packets pass unmodified through the client-facing server. Note that client-facing servers cannot inspect any packet payloads except for Initial packets.

[4.2.](#) Back-End Server

Back-end servers MUST have an up-to-date copy of the `ECHConfigList` the client-facing server is using, though it need not hold the private key, in order to properly process and generate the relevant transport parameters.

Back-end servers MUST NOT take any action based on the presence or contents of the `initial_encryption_context` transport parameter, as the client-facing server has likely stripped the Encryption Context out of the header.

In all other respects, they operate as Protected Initial servers.

[5.](#) 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 QUICv1 can also operate over this version of QUIC.

[6.](#) Security and Privacy Considerations

Sections [10.2](#), [10.3](#), [10.4](#), and [10.6](#) of [\[ECHO\]](#) apply to QUIC Protected

Initials as well.

Section 7.8 of [[VERSION-ALIASING](#)] is also applicable.

[6.1.](#) Forcing Downgrade

An attacker attempts to force a client to send an Initial that uses unprotected Initials by injecting a Version Negotiation packet (which implies the server no longer supports Protected Initials) or a Fallback packet (which implies the server has a new cryptographic context).

The weak form of this attack observes the Initial and injects the Version Negotiation or Fallback packet, but cannot drop the Initial. To counteract this, a client SHOULD NOT respond to these packets until they have waited for Probe Timeout (PTO) for a valid server Initial to arrive.

The strong form features an attacker that can drop Initial packets. In this case, the client can either abandon the connection attempt or connect with a unprotected Initial: using the fallback salt in response to a Fallback packet, or a different version in response to Version Negotiation.

If connecting with an unprotected Initial, the client MAY alter it to sanitize sensitive information and obtain either a correct ECHConfig or an aliased version before proceeding with its true connection attempt. However, the client Initial MUST lead to the authentication of a domain name the client trusts to provide accurate cryptographic information (usually the public_name from the ECHConfig). Advice for the Outer ClientHello in Section 10.5 of [[ECHO](#)] applies here.

Furthermore, if it received a Fallback packet, the client sends a public_key_failed transport parameter to detect the downgrade attack, and the server will terminate the connection if the Fallback packet

was an attack.

If the client received a Version Negotiation packet, it MUST implement a downgrade detection mechanism such as [\[I-D.ietf-quic-version-negotiation\]](#) or abandon the connection attempt. If it subsequently detects a downgrade detection, or discovers that the server does not support the same mechanism, it terminates the connection attempt.

Note that ECH is able to retrieve an up-to-date cryptographic context in 1 RTT, because the client hello has enough plaintext information to begin a handshake with the public_name. Protected Initials require reconnection with the public name, incurring a 2 RTT penalty to achieve the same result.

[6.2.](#) Initial Packet Injection

QUIC version 1 handshakes are vulnerable to DoS from observers for the short interval that endpoints keep Initial keys (usually ~1.5 RTTS), since Initial Packets are not authenticated. With version aliasing, attackers do not have the necessary keys to launch such an attack.

QUIC version 1 can use a fixed symmetric cipher for its Initial Packets because the encryption is not providing true security. As this design aspires to stonger guarantees, the Encryption Context field of the Initial header provides the codepoints to enable use of other symmetric ciphers should AES-128-GCM be compromised in the future. This is to provide cryptographic agility in accordance with [\[RFC7696\]](#).

[6.3.](#) Retry Injection

This version of QUIC does not improve the security of Retry packets with respect to QUIC version 1. The Retry Integrity Tag uses a well-

known key and relies on data in the Initial that triggered the Retry. It therefore protects against transmission errors and injection of Retry packets by off-path attackers that cannot observe the Initial. To detect Retry packets injected by observers, it relies on the subsequent exchange of transport parameters.

An attacker that consistently injects Retry packets in front of a server that also consistently sends Retry can result in a Denial of Service, as clients cannot accept two Retries in the same connection.

An alternate design would use the shared secret derived from the Client Initial Packet to generate keys for the Retry Integrity Tag, which would allow the client to immediately discard Retries injected by other parties. Unfortunately, this would require servers to perform an asymmetric crypto operation to send a Retry packet, when in a state where it is likely computationally limited.

It is possible to enhance the security of Retry by assuming this added computational cost. Such a design could also eliminate the complexity associated with adding an arbitrary value to the Packet Length field. The purpose of this addition is to avoid trial decryption to verify the configuration is correct, but this cost is negligible compared to extracting the shared secret.

[7.](#) IANA Considerations

[7.1.](#) QUIC Version Registry

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

Value: TBD

Status: permanent

Specification: This document

Change Controller: IETF

Contact: QUIC WG

7.2. QUIC Transport Parameter Registry

This document requests that IANA add the following three entries to the QUIC Transport Parameters registry:

Value	Parameter Name	Specification
TBD	public_key_failed	This document
TBD	ECHConfig	This document
TBD	initial_encryption_context	This document

Table 2

7.3. QUIC Transport Error Codes Registry

This document requests that IANA add the following entry to the QUIC Transport Error Codes registry:

Value: TBD

Code: INVALID_PROTECTED_INITIAL_DOWNGRADE

Description: Attacker is forcing insecure Initial

Specification: This document

8. References

8.1. Normative References

- [ECHO] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, [draft-ietf-tls-esni-14](https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14), 13 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14>>.
- [HPKE] Barnes, R. L., Bhargavan, K., Lipp, B., and C. A. Wood, "Hybrid Public Key Encryption", Work in Progress, Internet-Draft, [draft-irtf-cfrg-hpke-12](https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hpke-12), 2 September 2021, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hpke-12>>.

Internet-Draft

protected-initial

April 2022

[I-D.ietf-quic-version-negotiation]

Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", Work in Progress, Internet-Draft, [draft-ietf-quic-version-negotiation-07](https://datatracker.ietf.org/doc/html/draft-ietf-quic-version-negotiation-07), 5 April 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-version-negotiation-07>>.

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

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", [RFC 9001](https://www.rfc-editor.org/rfc/rfc9001), DOI 10.17487/RFC9001, May 2021, <<https://www.rfc-editor.org/rfc/rfc9001>>.

[8.2.](#) Informative References

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

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

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", [BCP 201](https://www.rfc-editor.org/rfc/rfc7696), [RFC 7696](https://www.rfc-editor.org/rfc/rfc7696), DOI 10.17487/RFC7696, November 2015, <<https://www.rfc-editor.org/rfc/rfc7696>>.

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

[VERSION-ALIASING]

Duke, M., "QUIC Version Aliasing", Work in Progress, Internet-Draft, [draft-duke-quic-version-aliasing-07](https://datatracker.ietf.org/doc/html/draft-duke-quic-version-aliasing-07), 25

October 2021, <<https://datatracker.ietf.org/doc/html/draft-duke-quic-version-aliasing-07>>.

Appendix A. Change Log

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

Duke & Schinazi

Expires 29 October 2022

[Page 17]

Internet-Draft

protected-initial

April 2022

A.1. since [draft-duke-quic-protected-initials-01](#)

- * Redesigned fallback again without Version Negotiation
- * Added the `initial_encryption_context` transport parameter

A.2. since [draft-duke-quic-protected-initials-00](#)

- * Additional text comparing ECH, Version Aliasing
- * Adapted to foreground the split-mode use case
- * Clarified server initials are encrypted
- * Retry keys are no longer generated from the shared secret
- * Complete Rewrite of Fallback
- * New transport parameters
- * Added crypto agility

Authors' Addresses

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

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,

United States of America
Email: dschinazi.ietf@gmail.com