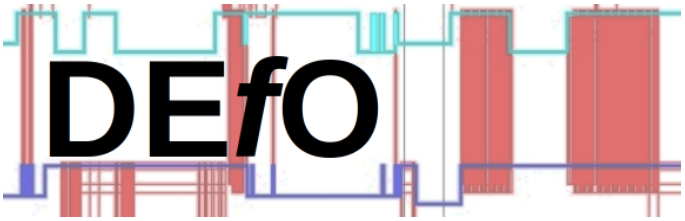

QUIC and ECH

A. Custura

Saturday 27th September, 2025

Abstract

Encrypted Client Hello (ECH) is a privacy-enhancement for the Transport Layer Security (TLS) protocol that underlies all web security and the security of many other Internet protocols. While the specification for ECH is relatively mature, (though not yet an Internet-RFC), and while implementations are already widespread, work remains to ensure that a random client and server can successfully use ECH.

QUIC is a transport protocol that enhances privacy and security for endpoints [1], deployed by several major Internet providers, including Google, Apple, and Cloudflare [2]. Standardised in 2021, QUIC is increasingly replacing web stacks using the traditional Transport Control Protocol (TCP) for web browsing.

To that end, this report describes issues with ECH-enabling QUIC as an input to future work on that topic.

Contents

Keywords: Encrypted Client Hello (ECH), QUIC

1 Introduction

Deployments of the TLS [3] protocol expose the name of the server (e.g. the web site DNS name) via the Server Name Indication (SNI) field in the first message sent (the ClientHello). The Encrypted Client Hello (ECH) [4] extension to TLS is a privacy-enhancing scheme that aims to address this leak.

This report discusses considerations for deploying ECH with the QUIC transport protocol, assuming the QUIC implementation relies on an open-source TLS library, such as OpenSSL.

The primary audience for this document are those implementing and deploying ECH, as well as those implementing QUIC. Secondarily, there may be lessons to learn for those designing protocols like ECH.

The Open Technology Fund (OTF, `https://www.opentech.fund/`) have funded the DEfO project (`https://defo.ie`) to develop ECH implementations for OpenSSL, and to otherwise encourage implementation and deployment of ECH. As we expect the implementation and deployment environment for ECH to change over time, this report will be updated as events warrant and is currently versioned based on the build-time of this PDF.

2 QUIC and ECH

In combination with HTTP, the TLS protocol provides security by encrypting the HTTP payload. Unlike HTTP versions 1 and 2, which operate over TCP (and may or may not use a TLS layer), HTTP version 3, standardised in 2022, is designed to use QUIC as a transport. Rather than forming a separate layer, TLS is tightly integrated with QUIC, and QUIC and TLS components are expected to cooperate, as illustrated in Figure 1.

QUIC can provide a reliable stream abstraction to applications in a similar way to TCP, and this is how it is expected to operate with TLS. RFC 9001 [5] describes in detail how TLS is used to secure the QUIC transport protocol.

Implementing ECH in the various QUIC software is expected to rely on enabling it in the underlying SSL library, similar to providing it for TCP, and then calling the relevant functions from the library on connection setup. Some differences exist, as QUIC also needs to configure TLS.

The interface between QUIC and TLS includes sending and receiving handshake messages [5]: a QUIC client requests TLS handshake bytes from the TLS implementation, while a QUIC server provides the TLS layer with the client's handshake bytes. QUIC carries this TLS handshake data in CRYPTO frames, directly over the QUIC transport, which substitutes the TLS record layer (required when transmitting data over TCP, as TCP does not have framing capabilities). In exchange, TLS provides QUIC with cryptographic keys and state change information.

QUIC can use 2 handshake modes:

- A 1 Round Trip Time (1-RTT) handshake, where application data is sent after the server has responded to the first handshake message from the client

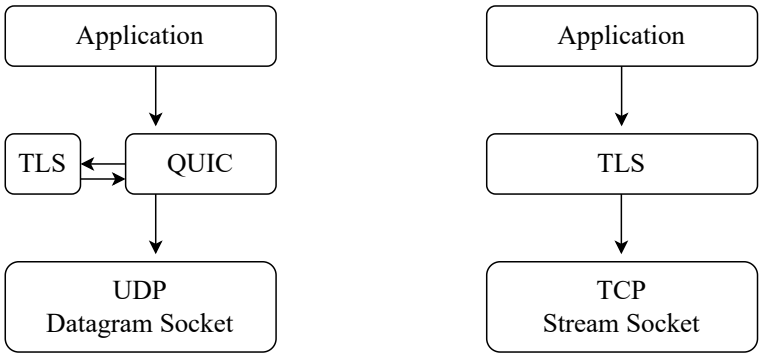

Figure 1: Compared to the strict layering of TLS and TCP, QUIC and TLS components are expected to cooperate.

- A 0-RTT handshake where application data is immediately sent, e.g. right after the client sends/server receives a TLS ClientHello

Once the TLS handshake is complete, QUIC can send application data. This is carried in Stream frames rather than in TLS Application Data records. TLS then does not send more data unless requested (e.g., as noted in [5], if the server wishes to provide additional or updated session tickets to a client).

3 Existing Implementations

Several open-source QUIC implementations exist. This report considers the Open Source client implementations that are actively evaluated by the QUIC interoperability runner [6], which performs automated tests to ensure the implementations are compatible with each other. Differences may arise where the standard has not given enough detail on a specific aspect of the protocol design, and this testing is expected to highlight differences in interpretation.

ECH support was determined from analysing the source code for these implementations in Table 1.

QUIC Impl.	QUIC Developer	TLS Library	ECH Support
`mvfst`	Facebook	fizz	Yes
`neqo`	Mozilla	NSS	Yes
`chrome`	Google	BoringSSL	Yes
`quiche`	Cloudflare	BoringSSL	No
`msquic`	Microsoft	OpenSSL	No
`xquic`	Alibaba	BoringSSL, BabaSSL	No
`s2n-quic`	AWS	rustls, s2n-tls	No
`lsquic`	LiteSpeed	BoringSSL	No
`ngtcp2`	T. Tsujikawa	BoringSSL	Yes
`picoquic`	C. Huitema	OpenSSL	No
`quic-go`	M. Seemann	Go Std. Lib.	No
`kwik`	P. Doornbosch	Agent15	No
`aioquic`	J. Lainé	OpenSSL	No
`quinn`	D. Ochtman, B.Saunders	rustls	No

Table 1: ECH Support across QUIC client implementations [6]

Before starting the handshake, QUIC also has to provide TLS with the transport parameters [1] that it wishes to carry. This is an additional requirement from TCP [1]. As per RFC 9001, the `quic_transport_parameters` extension has to be carried in a TLS ClientHello and the EncryptedExtensions messages during the handshake. Differences regarding transport parameter handling exist within the four QUIC implementations that were found to support ECH, with evidence that `neqo` allows different transport parameters in the inner and outer ClientHello, however cautioning against using this feature unless the ECH configuration has been validated, due to a mechanism that filters sensitive transport parameters in the outer ClientHello (see source code). For the other implementations (`mvfst`, `chrome`, and `ngtcp2`), this behaviour appears to depend on the underlying TLS library, however no mechanism for filtering sensitive transport parameters was noted in the libraries BoringSSL and fizz.

Some content carried in transport parameters could potentially benefit from being protected by ECH encryption. Determining which parameters, and under what circumstances, should not be carried in an Outer ClientHello will require further analysis best undertaken during implementation. For example, `neqo` retains only the parameters necessary to ensure the connection attempt succeeds, as well as any that might affect connection configuration in ways that could impact operation[2].

4 Considerations Arising

The interface between QUIC and TLS also includes any functions required to configure TLS, and [5] notes this step establishes whether the QUIC or TLS implementation is responsible for peer validation (see Section 4.1 of [5]). When using ECH, the name that should be validated is present on the inner ClientHello and so the library used must be aware of this to avoid a certificate for the name in the outer ClientHello being considered valid. Especially for the circumvention use case, the name used in the outer Hello may be deliberately chosen as not one under the server operator's control to take advantage of collateral freedom.

Some consideration should also be given to how QUIC transport parameters are to be used within the outer or inner ClientHello when using ECH, and whether any filtering mechanism exists for these in the underlying TLS library that could interact with any technique for falling back to a non-ECH connection.

Finally, the QUIC initial handshake message can also establish peer address validation, and in some cases a server must parse the complete ClientHello before deciding whether to accept a new QUIC connection. The use of ECH with post-quantum encryption can cause ClientHello messages to exceed the size of a QUIC Initial packet (minimum 1200 Bytes) as well as a standard Ethernet packet (1500 Bytes). If that ClientHello is split across multiple Initial packets, the server would need to buffer the received fragments. Fragmentation creates a potential attack surface, as an unvalidated client address could force the server to consume excessive resources. QUIC implementers enabling ECH should be aware that where ClientHello messages span multiple packets, servers may refuse to buffer them for reassembly to prevent possible abuse [5]. To mitigate this issue, an implementer could use server-side "Retry" packets for address validation, as described in Section 8.1 of [1]. These packets contain a token that the client can include in later packets to confirm its address.

5 Conclusions

The report concludes that there are no current obstacles to adopting ECH across QUIC implementations, with several mature implementations already providing support. However, developers must carefully consider peer validation, transport parameter handling, and ClientHello size.

[1] `https://www.iana.org/assignments/quic/quic.xhtml`

[2] These include `original_destination_connection_id`, `stateless_reset_token`, `initial_source_connection_id`, `retry_source_connection_id` and `version_information`. Other parameters are included (`ack_delay_exponent`, `max_ack_delay`, and `max_udp_payload_size`) on the basis they pose no privacy risk (see source code)

Acknowledgements

Thanks to the Open Technology Fund for ongoing funding of the DEfO project. (And for their patience while the IETF process for ECH takes... ages;-) Thanks in particular to the people working on DEfO who contributed to this work including: Kerry Hartnett, John Hess, Iain Learmonth, Niall O'Reilly, Jochen Sprickerhof and Hans-Christoph Steiner. Thanks also to the developers of other ECH implementations (including NSS, boringssl, wolfssl, golang and rustls) for co-operating as we worked on interoperability, and to the maintainers of OpenSSL, curl, lighttpd and haproxy for ECH related discussions and PR-processing along the way.

All errors and omissions of course remain the fault of the author.

References

[1] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the ACM SIGGCOM Conference*. ACM, pp. 183–196.

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[4] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood, "TLS Encrypted Client Hello," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-esni-22, Sep. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-esni/22/

[5] M. Thomson and S. Turner, "Using TLS to Secure QUIC," RFC 9001, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9001

[6] M. Seemann. (2021) QUIC Interop Runner. [Online]. Available: https://interop.seemann.io