Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) vs. QUIC

Shan Chen[1], Samuel Jero[2], Matthew Jagielski[3],
Alexandra Boldyreva[1], and Cristina Nita-Rotaru[3]

[1] Georgia Institute of Technology {shanchen,sasha}@gatech.edu
[2] Purdue University sjero@sjero.net
[3] Northeastern University jagielski.m@husky.neu.edu, c.nitarotaru@neu.edu

Abstract. Secure channel establishment protocols such as TLS are some of the most important cryptographic protocols, enabling the encryption of Internet traffic. Reducing the latency (the number of interactions between parties) in such protocols has become an important design goal to improve user experience. The most important protocols addressing this goal are TLS 1.3 over TCP Fast Open (TFO), Google's QUIC over UDP, and QUIC[TLS] (a new design for QUIC that uses TLS 1.3 key exchange) over UDP. There have been a number of formal security analyses for TLS 1.3 and QUIC, but their security, when layered with their underlying transport protocols, cannot be easily compared. Our work is the first to thoroughly compare the security and availability properties of these protocols. Towards this goal, we develop novel security models that permit "layered" security analysis. In addition to the standard goals of server authentication and data privacy and integrity, we consider the goals of IP spoofing prevention, key exchange packet integrity, secure channel header integrity, and reset authentication, which capture a range of practical threats not usually taken into account by existing security models that focus mainly on the crypto cores of the protocols. Equipped with our new models we provide a detailed comparison of the above three protocols. We hope that our results will help protocol designers in their future protocol analyses and practitioners to better understand the advantages and limitations of novel secure channel establishment protocols.

Keywords: applied cryptography · provable security · TLS · QUIC · secure channel · availability · network protocols

1 Introduction

Motivation. Nowadays, more than half of all Internet traffic is encrypted according to a 2017 EFF report [20], with Google reporting that 93% of its traffic is encrypted as of January 2019 [1]. This widespread Internet traffic encryption is enabled by protocols that allow two parties (where one or both parties have a public key certificate) to establish a secure communication channel over the insecure Internet. Typically, the parties first authenticate all parties holding a public key certificate and agree on a session key — the key exchange phase.

Then, this session key is used to encrypt the communication during the session — the secure channel phase. We will refer to such protocols as secure channel establishment protocols.

The main secure channel establishment protocol in use today is TLS. The session key establishment with TLS today involves 3 round-trip times (RTTs) of end-to-end communication, including the cost of establishing a TCP connection before the TLS connection. Further, this TCP cost is paid every time the two parties communicate with each other, even if the connection is interrupted and then immediately resumed. Given that most encrypted traffic is web traffic, this cost represents a significant performance bottleneck, a nuisance to users, and financial loss to companies. For instance, back in 2006 Amazon found that every 100ms of latency cost them 1% in sales [34], while a typical RTT on a connection from New York to London is 70ms [22].

Not surprisingly, many efforts in recent years have focused on reducing latency in secure channel establishment protocols. The focus has been on reducing the number of interactions (or RTTs) during session establishment and resumption without sacrificing much security. The most important protocols addressing this goal are TLS 1.3 [43] (the just-released successor to the current TLS 1.2 standard) and Google's QUIC [45].

With TLS 1.3, it is possible to reduce the number of RTTs (prior to sending encrypted data) during session resumption to 1, by utilizing a session ticket that was saved during a previous communication. The remaining 1-RTT during session resumption is due to the aforementioned TCP connection. However, one recent optimization for TCP, called TCP Fast Open (TFO) [42,10] extends TCP to allow for 0-RTT resumption connections, so that the client may begin data transmission immediately. The mechanism underlying this optimization is a cookie saved from previous communication, similar to the ticket used by TLS 1.3.

Like TLS 1.3, Google's QUIC uses weaker initial keys, under which data can be encrypted earlier, and a token saved from previous communication between the parties. But unlike TLS, QUIC operates over UDP rather than TCP. Instead of relying on TCP for reliability, flow control, and congestion control, QUIC implements its own data transmission functionality, integrating connection establishment with key exchange. These features allow QUIC to have 1-RTT full connections and 0-RTT resumption connections.

In addition to TLS 1.3 over TFO and QUIC over UDP, there is a new design for QUIC [23] (which we refer to as QUIC[TLS] [47] to indicate that it borrows the key exchange from TLS 1.3) over UDP. These 3 protocols win in terms of the number of interactions, but how does their security compare?

At first glance, the question is easy to answer. Recent works have done formal security analyses of TLS 1.3 [28,5,14,11,29,15,33,18,13,4,12,6] and Google's QUIC [17,35]. Most works confirm that (the cryptographic cores of) both protocols are provably secure under reasonable computational assumptions. Moreover, as shown in [35,18], their 0-RTT data transmission designs cannot achieve the same strong security guaranteed by classical key exchange protocols with at least

one RTT. In particular, the 0-RTT keys do not provide forward secrecy and the 0-RTT data suffers from replay attacks. Overall, it might seem that all three layered protocols mentioned above are equally secure.

However, a closer look reveals that the answer is not that simple. First, all aforementioned formal security analyses, except for [35] analyzing the IP spoofing (source validation) of QUIC, did not consider packet-level availability attacks. Therefore, it is not clear at the packet level what security can be achieved and what attacks can be prevented by these protocols. In other words, we have no formal understanding of what security can be obtained when layering protocols. We note that for protocols targeting low latency availability is essential, and since it can be assured to some degree by cryptographic means, a cryptographic analysis is very important. Also, TFO uses some cryptographic primitives, such as a cookie, to prevent IP spoofing, but, to the best of our knowledge, no formal analysis has been done. Furthermore, the security of QUIC[TLS] has not been formally analyzed (although some security aspects can be reduced to those of Google's QUIC and TLS 1.3).

Our contributions. The goal of our work is to help public understanding of how security compares for the most latency-efficient secure channel establishment protocols on the market today. By including packet-level attacks in our analysis, our results also shed light on how the reliability, flow control, and congestion control of both approaches compare, in adversarial settings.

To compare security, we first need to define a general protocol syntax for secure channel establishment and fix a security model for it. We take *Quick Connections (QC)* protocol definition [35] as our starting point. To accommodate protocol syntaxes of TLS 1.3 and QUIC[TLS], we extend the QC protocol to a more general *Multi-Stage Authenticated and Confidential Channel Establishment (msACCE)* protocol, which allows more keys to be set during each session. The details are in Section 4.1.

Then, we extend the *Quick Authenticated and Confidential Channel Establishment (QACCE)* security model [35] to two msACCE security models — msACCE-std and msACCE-pauth — that are general enough for all layered secure channel establishment protocols mentioned above. The former model, msACCE-std, is fairly standard and is for core cryptographic security. The latter model, msACCE-pauth, is novel and is for packet-level security. For this packet-authentication model we extend the definition of IP-Spoofing Prevention from [35], and also define Key Exchange (KE) Header Integrity, KE Payload Integrity, Secure Channel (SC) Header Integrity, and Reset Authentication.

Equipped with our new models (see [9] and Section 4.2 for details), we study the security and availability functionalities provided by TFO+TLS 1.3, UDP+QUIC, and UDP+QUIC[TLS]. We first confirm that all protocols provably satisfy the standard security notions of Server Authentication and Channel Security given that their building blocks are secure. The results mostly follow from prior works and we just have to argue that they still hold for our msACCE-std security model (which is an extension to previous models). Due to lack of space, we treat the above standard security notions and corresponding proto-

col security analyses in the full version [9], and here we focus on the novel packet-level security. We analyze the first 2 low latency protocols under our new model in Section 5 and refer to the full version [9] for the security analysis of UDP+QUIC[TLS]. Some of our theoretical findings capture practical availability attacks that the networking community has been slowly uncovering via manual investigation over the last 30 years [46,27,2,8,31,30,25,41,7,21,40,36,48,26], such as TCP flow control manipulation, TCP acknowledgment injection, etc. Our findings also discover new weaknesses (e.g., those that allow manipulating the early key exchange packets without being detected by the communicating parties). Furthermore, our results prove security guarantees for certain goals (such as showing that TFO's cookie mechanism provably achieves the security goal of IP Spoofing Prevention and QUIC[TLS]'s stateless reset mechanism provably achieves the security goal of Reset Authentication). Table 1 in Section 5 summarizes our results.

2 Background

Network protocols are designed following a layered network stack model where each layer has its own functionality, defines an interface for use by higher layers, and relies only on the properties of lower layers. In this work, we are concerned with three layers: network, represented by the IP protocol; transport, represented by UDP and TCP with the Fast Open optimization (TFO); and application, represented by TLS or QUIC.

TCP Fast Open. TCP Fast Open (TFO) is an optimization which introduces a simple modification to the TCP connection establishment handshake to reduce the 1-RTT connection establishment latency of TCP and allow for 0-RTT handshakes. The mechanism through which 0-RTT is achieved is a cookie that is obtained by the client first time it communicates with a server and cached for later uses. This cookie is intended to prevent replay attacks while avoiding the need for servers to keep expensive state. It is generated by the server, authenticates client IP address, and has a limited lifetime. Generation and verification have low overhead. Cookies are sent in the TFO option field in SYN packets (see Fig. 1 for details).

TLS 1.3. The recently standardized TLS 1.3 [43] improves TLS 1.2. Most relevant, it enables 0-RTT handshakes at the TLS level. In a TLS 1.3 full connection (see Fig. 1, fourth message), the client begins by sending a `ClientHello` message containing a list of ciphersuites to use with key shares for each. The server responds with a `ServerHello` message containing the ciphersuite to use and its key share. At this point, an initial encryption key is derived and all future messages are encrypted. The server also sends an `EncryptedExtensions` message containing any extension data, a `CertificateRequest` message if doing client authentication, a `ServerCertificate` message containing the server's certificate, a `ServerCertificateVerify` message containing a signature over the handshake with the private key corresponding to the server's certificate, and a `ServerFinished` message containing an HMAC of all messages in the handshake.

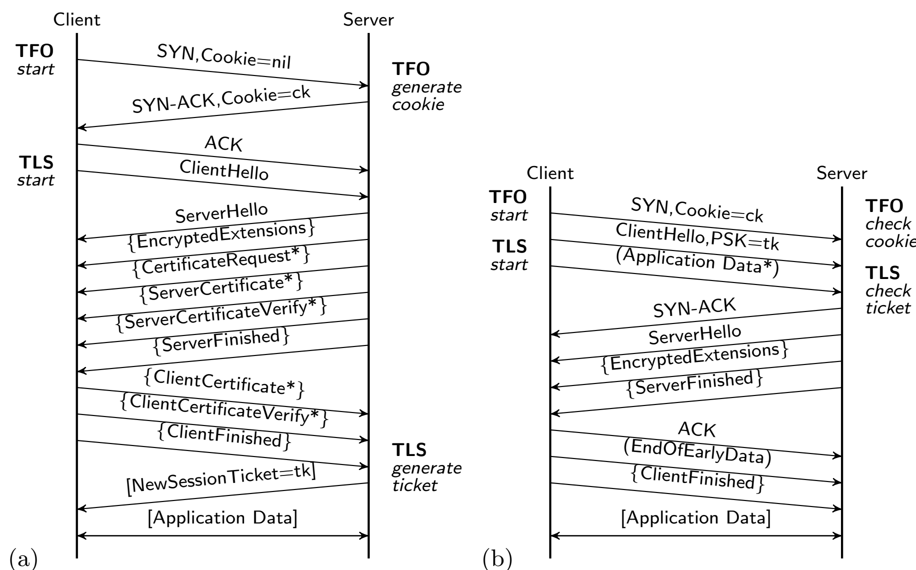

Fig. 1. TFO+TLS 1.3 (EC) DHE 2-RTT full handshake (a) and TFO+TLS 1.3 PSK-(EC) DHE 0-RTT resumption handshake(b). * indicates optional messages. () indicates messages protected using the 0-RTT keys derived from a pre-shared key. {} and [] indicate messages protected with initial and final keys.

The client receives these messages, verifies their contents, and responds with `ClientCertificate` and `ClientCertificateVerify` messages if doing client authentication before finishing with a `ClientFinished` message containing an HMAC of all messages in the handshake. At this point, a final encryption key is derived and used for encrypting all future messages. If the server supports 0-RTT connections, one final handshake message, the `NewSessionTicket` message, will be sent by the server to provide the client with an opaque session ticket to be used in a resumption session.

In later TLS 1.3 resumption connections to this server, the client uses the session ticket established in the prior full connection to do a 0-RTT connection. In this case, the client sends a `ClientHello` message indicating a pre-shared-key ciphersuite, a ciphersuite to use for the final key, and the cached session ticket. The client can then derive an encryption key and begin sending 0-RTT data. The server will verify the session ticket, use it to establish the same encryption key, and send a `ServerHello` message containing the ciphersuite to use and its final key share. At this point, an initial encryption key is derived and all future messages are encrypted. The server also sends an `EncryptedExtensions` message containing any extension data and a `ServerFinished` message containing an HMAC of all messages in the handshake. The client receives these messages, verifies their contents, and responds with an `EndOfEarlyData` message and a `ClientFinished` message containing an HMAC of all messages in the handshake.

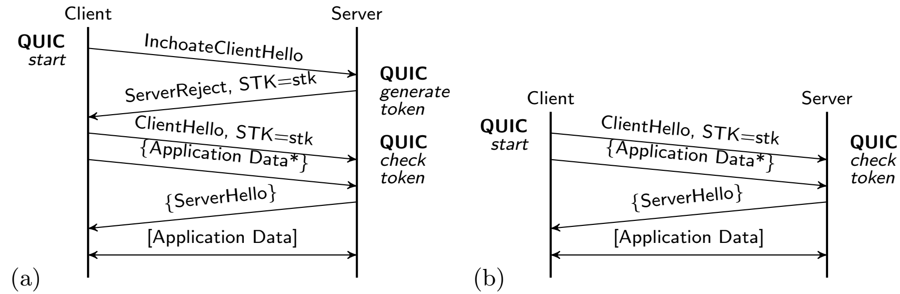

Fig. 2. QUIC 1-RTT full handshake (a) and UDP+QUIC 0-RTT resumption handshake (b). * indicates optional messages. {} and [] indicate messages protected with initial and final keys.

At this point, a final encryption key is derived and used for encrypting all future messages.

TLS 1.3 over TFO. Layering TLS 1.3 over TCP Fast Open enables true 0-RTT connections. In a full connection to a TFO+TLS 1.3 server, the client requests a TFO cookie in the TCP SYN and then does a full TLS 1.3 handshake once the TCP connection completes. This takes 3-RTTs (see Fig. 1), but provides a cached TFO cookie and cached TLS session ticket. In subsequent resumption connections to this server, the client can use the TFO cookie to establish a 0-RTT TCP connection and include the TLS 1.3 `ClientHello` message in the SYN packet. The TLS `ClientHello` message can use the cached TLS session ticket to perform a 0-RTT resumption handshake. Thus, the TCP and TLS 1.3 connections are established at the same time, as shown in Fig. 1.

QUIC over UDP. Quick UDP Internet Connections (QUIC) is a transport protocol developed by Google and implemented by Chrome and Google servers since 2013 [45]. QUIC provides a very similar set of services to TFO+TLS 1.3, however instead of modifying TCP to enable 0-RTT connection establishment, QUIC replaces TCP entirely, using UDP.

QUIC packets contain a public header and a set of frames that are encrypted and authenticated after initial connection setup. The header contains a set of public flags, a unique 64bit connection identifier referred to as `cid`, and a variable length packet number. All other protocol information is carried in control and stream (data) frames that are encrypted and authenticated.

To provide 0-RTT, QUIC caches information about the server that will enable the client to determine the encryption key to be used for each new connection. As shown in Fig. 2, the first time a client contacts a given server it sends an empty (Inchoate) `ClientHello` message. The server responds with a `ServerReject` message containing the server's certificate, an object called an `scfg`, (contains a variety of information about the server, including a Diffie-Hellman share from the server), supported encryption and signing algorithms, and flow control parameters. Along with the `scfg`, the server sends the client a source-address token

or stk. The stk is used to prevent IP spoofing. It contains an encrypted version of the client's IP address and a timestamp.

With this cached information, a client can establish an encrypted connection with the server. It first ensures that the scfg is correctly signed by the server's certificate which is valid and then sends a ClientHello indicating the scfg its using, the stk value it has cached, a Diffie-Hellman share for the client, and a client nonce. After sending the ClientHello, the client can create an initial encryption key and send additional encrypted Application Data packets. In fact, to take advantage of the 0-RTT connection establishment it must do so. When the server receives the ClientHello message, it validates the stk and client nonce parameters and creates the same encryption key using the server share from the scfg and the client's share from the ClientHello message.

At this point, while both client and server have established the connection, setup encryption keys and all further communication between the parties is encrypted, the connection is not forward secure yet, meaning that compromising the server would compromise all previous communication because the server's Diffie-Hellman share is the same for all connections using the same scfg. To provide forward secrecy for all data after the first RTT, the server sends a ServerHello message after receiving the client's ClientHello which contains a newly generated Diffie-Hellman share. Once the client receives this message, client and server derive and begin using the new forward secure encryption key.

For the client that has connected to a server before, it can instead initiate a resumption connection. This consists of only the last two steps of a full connection, sending the ClientHello and ServerHello messages as shown in Fig. 2.

QUIC with TLS 1.3 Key Exchange over UDP. A new version of QUIC [23], which also supports 0-RTT, describes several improvements of the previous design. The most important change is replacing QUIC's key exchange with the one from TLS 1.3, as specified in the latest Internet draft [47]. We provide more details (e.g., about its new stateless reset feature) in the full version [9].

3 Preliminaries

Public Key Infrastructure. For simplicity, we assume the public keys used in our analysis are supported by a *public key infrastructure (PKI)* and do not consider certificates or certificate checks explicitly. In other words, we assume each public key is certified and bound to the corresponding party's identity.

PRF and AEAD. In the full version [9] we recall the security definitions of a *pseudorandom function (PRF)* F and a stateful *authenticated encryption with associated data (AEAD)* scheme sAEAD. Accordingly, there we provide the definitions for the corresponding advantages: $\mathbf{Adv}_F^{\mathsf{prf}}(A), \mathbf{Adv}_{\mathsf{sAEAD}}^{\mathsf{aead}}(A)$. We also refer to [44] for the syntax and security definitions of a nonce-based AEAD scheme.

4 msACCE Protocol and its Security

In this section, we define the syntax and two security models for *Multi-Stage Authenticated and Confidential Channel Establishment (msACCE)* protocols.

4.1 Protocol Syntax

Our msACCE protocol is an extension to the *Quick Connection (QC)* protocol proposed by Lychev *et al.* [35] and the *Multi-Stage Key Exchange (MSKE)* protocol proposed by Fischlin and Günther [17] (and further developed by [14,15,33,18]). Even though the authors of [35] claimed their QC protocol syntax to be general, TLS 1.3 does not fit it well because TLS 1.3 has two initial keys and one final key in 0-RTT resumption while QC captures only one initial key. On the other hand, the MSKE protocol and its extensions focus only on the key exchange phases.

Our msACCE protocol syntax inherits many parts of the QC protocol syntax but extends it to a multi-stage structure and additionally covers session resumptions (explicitly, unlike QC), session resets, and header-only packets exchanged in secure channel phases. The detailed protocol syntax is defined below.

A msACCE protocol is an interactive protocol between a client and a server. They establish keys in one or more stages and exchange messages encrypted and decrypted with these keys. Messages are exchanged via *packets*. A packet consists of source and destination IP addresses[4] $\mathrm{IP}_{\mathsf{s}}, \mathrm{IP}_{\mathsf{d}} \in \{0,1\}^{32} \cup \{0,1\}^{64}$, a header, and a payload. Each party P has a unique IP address IP_P.

The protocol is associated with the security parameter $\lambda \in \mathbb{N}_+$, a key generation algorithm Kg that takes as input 1^λ and outputs a public and secret key pair, a header space[5] (for transport and application layers) $\mathcal{H} \subseteq \{0,1\}^*$, a payload space $\mathcal{PD} \subseteq \{0,1\}^*$, header and payload spaces $\mathcal{H}_{\mathsf{rst}} \subseteq \mathcal{H}, \mathcal{PD}_{\mathsf{rst}} \subseteq \mathcal{PD}$ for reset packets (described later), a resumption state space $\mathcal{RS} \subseteq \{0,1\}^*$, a stateful AEAD scheme[6] $\mathsf{sAEAD} = (\mathsf{sG}, \mathsf{sE}, \mathsf{sD})$ (with a key space $\mathcal{K} = \{0,1\}^\lambda$, a message space $\mathcal{M} \subseteq \{0,1\}^*$, an associated data space $\mathcal{AD} \subseteq \{0,1\}^*$, and a state space $\mathcal{ST} \subseteq \{0,1\}^*$), *disjoint*[7] message spaces $\mathcal{M}_{\mathsf{KE}}, \mathcal{M}_{\mathsf{SC}}, \mathcal{M}_{\mathsf{pRST}} \subseteq \mathcal{M}$ with $\mathcal{M}_{\mathsf{KE}}, \mathcal{M}_{\mathsf{SC}}$ for messages encrypted during key exchange and secure channel phases respectively and $\mathcal{M}_{\mathsf{pRST}}$ for pre-reset messages (described later) encrypted in a secure channel phase, a server configuration generation function $\mathsf{scfg_gen}$ described below.

---

[4] For the network-layer protocols, we only consider the Internet Protocol and its IP address header fields because our model mainly focuses on the application and transport layers and additionally only captures the IP-spoofing attack.

[5] Some protocol header fields (e.g., port numbers, checksums, etc.) can be excluded if they are not the focus of the security analysis.

[6] To fit TLS 1.3's encryption scheme, unlike QACCE we model QUIC's encryption scheme as a more general stateful AEAD scheme rather than a nonce-based one.

[7] Disjointness is a reasonable assumption as practical protocols (such as the 3 layered protocols that we consider) enforce different leading bits for different types of messages.

The protocol's execution is associated with the universal notion of time divided into discrete periods $\tau_1, \tau_2, \ldots$. During its execution, both parties can keep states that are initialized to the empty string ε. In the beginning of each time period, the protocol may periodically update each server's configuration state $\texttt{scfg}$ with $\mathsf{scfg_gen}$ (which takes as input 1^λ, a server secret key, and a time period, then outputs a server configuration state). Otherwise, $\mathsf{scfg_gen}$ is undefined and without loss of generality the protocol is executed within a single time period.

A *reset* packet enables a sender, who lost its session state due to some error condition (e.g., server reboots, denial-of-service attacks, etc.), to abruptly terminate a session with the receiver. A *pre-reset* message (e.g., a reset token in QUIC[TLS]) is sent to the receiver in a secure channel phase[8] before the sender loses its state in order to authenticate the sender's reset packet. Each session has *at most one* pre-reset message for each party. A *non-reset* packet is not a reset packet. A *header-only* packet has no payload.

We say a party *rejects* a packet if its processing the packet leads to an error (defined according to the protocol), and *accepts* it otherwise.

The protocol has two modes, *full* and *resumption*. Its corresponding executions are referred to as the full and resumption sessions. Each resumption session is associated with a *single* previous full session and we say the resumption session *resumes* its associated full session. In the beginning of a full or resumption session, each party takes as input a list of messages[9] $\mathcal{M}^{\mathsf{snd}} = (M_1, \ldots, M_l), M_i \in \mathcal{M}_{\mathsf{SC}}, l \in \mathbb{N}$ (where the total message length $|\mathcal{M}^{\mathsf{snd}}|$ is polynomial in λ and $\mathcal{M}^{\mathsf{snd}}$ can be empty) as well as the other party's IP address. In a full session, the server runs $\mathsf{Kg}(1^\lambda)$ to generate a public and secret key pair and sends its public key to the client as input. In a resumption session, each party additionally takes as input its own resumption state $rs \in \mathcal{RS}$ (set in the associated full session). In either case, the client sends the first packet to start the session.

A D-stage msACCE protocol consists of $D \in \mathbb{N}_+$ successive stages and each stage, e.g., the d-th ($d \in [D]$) stage, consists of one or two phases described as follows:

1) Key Exchange. At the end of this phase each party sets its d-th stage key $k^d = (k_c^d, k_s^d)$. At most one of k_c^d and k_s^d can be $\perp$, i.e., unused.[10] If this is the final stage in a full session, each party can send additional messages[11] in $\mathcal{M}_{\mathsf{KE}}$ encrypted with k^d and by the end of this phase each party sets its own resumption state.

---

[8] A pre-reset message can also be carried within an *encrypted* key exchange packet. We consider it encrypted as a separate secure channel packet to get a clean packet-authentication security model described later.

[9] For simplicity, we consider transportation of *atomic* messages rather than a data *stream* that can be modeled as a stream-based channel [19] and later extended to capture multiplexing [37].

[10] This captures the case where a 0-RTT key only consists of a client encryption key while the server encryption key does not exist.

[11] This captures the post-handshake key exchange messages that are used for session resumption, post-handshake authentication, key update, etc.

2) Secure Channel. This phase is mandatory for the final stage but optional for other stages. In this phase, the parties can exchange messages from their input lists as well as pre-reset messages, encrypted and decrypted using the associated stateful AEAD scheme with k^d. The client uses k_c^d to encrypt and the server uses it to decrypt, whereas the server uses k_s^d to encrypt and the client uses it to decrypt. They may also send reset or header-only packets. At the end of this phase, each party outputs a list of received messages (which may be empty) $\mathcal{M}_i^{\mathsf{rcv}} = (M'_1, \ldots, M'_{l'_i})$, $l'_i \in \mathbb{N}$, $M'_i \in \mathcal{M}_{\mathsf{SC}}$.

Each message exchanged between the parties must belong to some unique phase at some unique stage. One stage's second phase and the next stage's first phase may overlap, and the two phases in the final stage may also overlap. We call the final stage key the *session* key and the other stage keys the *interim* keys.

Correctness. Consider a client and a server running a D-stage msACCE protocol in either mode without sending any reset packet. Each party's input message list $\mathcal{M}^{\mathsf{snd}}$, in which the messages are sent among D stages according to any partitioning $\mathcal{M}^{\mathsf{snd}} = \mathcal{M}_1^{\mathsf{snd}}, \ldots, \mathcal{M}_D^{\mathsf{snd}}$, is equal to the other party's total output message list $\mathcal{M}^{\mathsf{rcv}} = \mathcal{M}_1^{\mathsf{rcv}}, \ldots, \mathcal{M}_D^{\mathsf{rcv}}$, in which the message order is preserved. Each party terminates its session upon receiving the other party's reset packet.

Remark. With our more general protocol syntax, the ACCE [24] and QC [35] protocols can be classified into 1-stage and 2-stage msACCE protocols respectively.

4.2 Security Models

We propose two security models respectively for basic authenticated and confidential channel security and packet authentication. Our models do not consider the key exchange and secure channel phases independently, as was the case for some previous QUIC and TLS 1.3 security analyses [17,14,15,33,18], because QUIC's key exchange and secure channel phases are inherently inseparable and the TLS 1.3 full handshake does not fit into a composability framework, as discussed in [35,15]. We refer to the full version [9] for our basic model (which we call msACCE-std) that considers standard security goals such as server authentication and channel security (which captures data privacy and integrity) for msACCE protocols. Here we only present our novel msACCE packet-authentication (msACCE-pauth) model.

msACCE-pauth Overview. In this model, we consider security goals related to packet authentication beyond those captured by the basic model. Note that msACCE-std essentially focuses only on the packet fields in the application layer, while msACCE-pauth further covers transport-layer headers and IP addresses.

First, we consider IP spoofing prevention (a.k.a. source authentication) as with the QACCE model, but, as illustrated later, generalize one of the QACCE queries to additionally capture IP spoofing attacks in the full sessions. Then we define four novel packet-level security notions (elaborated later): *KE Header Integrity*, *KE Payload Integrity*, *SC Header Integrity*, and *Reset Authentication*,

which enable a comprehensive and fine-grained security analysis of layered protocols.

In particular, KE Header and Payload Integrity respectively capture the header and payload integrity of key exchange packets. Such security issues have not been investigated before and, as we show later, lead to new availability attacks for both TFO+TLS 1.3 and UDP+QUIC. Furthermore, we employ SC Header Integrity to capture the header integrity of non-reset packets in secure channel phases. Note that, unlike the availability attacks shown in [35], successful attacks breaking our security notions are *harder or impossible to detect* by the client as they do not affect the client's session key establishment, so they are more harmful in this sense. Finally, our model captures malicious *undetectable* session resets in a secure channel phase with Reset Authentication.

msACCE-pauth Definitions. Like previous models, we consider a very powerful adversary who can control communications between honest parties, can adaptively learn their stage keys, and can adaptively corrupt servers to learn their long-term keys and secret states. Our detailed security model is defined below.

Protocol Entities. The set of parties $\mathcal{P}$ consists of two disjoint type of parties: clients $\mathcal{C}$ and servers $\mathcal{S}$, i.e., $|\mathcal{P}| = |\mathcal{C}| + |\mathcal{S}|$.

Session Oracles. To capture multiple sequential and parallel protocol executions, each party $P \in \mathcal{P}$ is associated with a set of session oracles $\pi_P^1, \pi_P^2, \ldots$, where π_P^i models P executing a protocol instance in session $i \in \mathbb{N}_+$.

Matching Conversations. As part of the security model, *matching conversations* are used to model entity authentication, session key confirmation, and handshake integrity. A client (resp. server) oracle has a matching conversation with a server (resp. client) oracle if and only if both session oracles observe the same[12] *session identifier* `sid` defined according to the protocol specifications and security goals. Note that a msACCE protocol may have two different session identifiers in full and resumption modes, but for simplicity we use the same notation `sid`. Compared to the general definition of matching conversations [3,24], `sid` is often defined as a *subset* of the whole communication transcript. For instance, QUIC's `sid` in QACCE [35] is defined as the second-round key exchange messages, i.e., `ClientHello` and `ServerHello`, while the first-round messages are excluded to allow for valid but different source-address tokens or signatures. Similarly, TLS 1.2's `sid` in ACCE [28] is defined as the first three key exchange messages, while the rest are excluded to allow for valid but different encrypted `Finished` messages.

[12] As discussed in [24], two session oracles having matching conversations with each other may not observe the same transcript due to the gap between one oracle sending a message and the other receiving it. We can use *symmetric* session identifiers to define matching conversations because our msACCE-std model focuses only on server authentication and we require session identifiers to exclude, if any, a client oracle's last key exchange message(s) sent immediately before it sets its session key.

Peers. We say a client oracle and a server oracle are each other's *peer* if they observe the same first-stage session identifier $\mathtt{sid}_1$ (i.e., $\mathtt{sid}$ restricted to the first stage), which intuitively means that they set the first stage key with each other. Note that a client oracle may have more than one peers if $\mathtt{sid}_1$ consists of only message(s) sent from the client oracle, which can be replayed to the same[13] server to establish multiple (identical) first-stage keys. Therefore, a session oracle's peer may not be its final unique communication partner. Instead, the real partner is the session oracle with which the oracle has a matching conversation.

Security Experiments. In the beginning of the experiments, run $\mathsf{Kg}(1^\lambda)$ for all servers to generate the public and secret key pairs and initialize the global states of all parties and the local states of all session oracles. In the beginning of each time period, run $\mathsf{scfg_gen}$ (if defined) for each server to update its configuration state $\mathtt{scfg}$. We assume that both the server oracles and the adversary A are aware of the current time period. Let $N \in \mathbb{N}_+$ denote the maximum number of msACCE protocol instances for each party. The adversary A is given all public keys and the IP addresses associated with all parties and then interacts with the session oracles via the same $\mathsf{Connect}, \mathsf{Resume}, \mathsf{Send}, \mathsf{Reveal}$, $\mathsf{Corrupt}$ queries as in the msACCE-std model[14] (which respectively give the adversary abilities to start and resume a session, send key exchange messages and get responses, reveal session keys, and corrupt servers, referring to the full version [9] for more details), as well as the following:

• $\mathsf{Connprivate}(\pi_C^i, \pi_S^j, \mathtt{cmp})$, for $C \in \mathcal{C}$, $S \in \mathcal{S}$, $i, j \in [N], \mathtt{cmp} \in \{0, 1\}$.
This query always returns $\perp$. If $\mathtt{cmp} = 1$, π_C^i and π_S^j establish a *complete* full session privately without showing their communication to the adversary. If $\mathtt{cmp} = 0$, π_C^i and π_S^j establish a *partial* full session privately such that the last packet sent from π_C^i right before π_S^j sets its first stage key is blocked.

This query allows the adversary to establish a complete or partial full session between any client and server oracles without observing their communication. By taking an additional flag $\mathtt{cmp}$ as input, this query extends the QACCE $\mathsf{Connprivate}$ query [35] to model IP-spoofing attacks happening in both *full* and resumption sessions.

• $\mathsf{Pack}(\pi_P^i, ad, m)$, for $P \in \mathcal{P}, i \in [N], ad \in \mathcal{AD}, m \in \mathcal{M}_{\mathsf{SC}} \cup \mathcal{M}_{\mathsf{pRST}} \cup \{\mathtt{prst}, \mathtt{rst}\}$.
This query returns $\perp$ if π_P^i is not in a secure channel phase. If $m \in \mathcal{M}_{\mathsf{SC}} \cup \mathcal{M}_{\mathsf{pRST}}$, it asks π_P^i to output the packet that it would send to its peer(s) for the specified associated data ad and message m according to the protocol, then returns this packet. If $m = \mathtt{prst}$, π_P^i generates its pre-reset message (if any, hidden from the adversary), encrypts it with the specified associated data ad, and outputs the resulting packet, then this packet is returned. (Recall that each oracle has at most one pre-reset message, so at most one input message $m \in \mathcal{M}_{\mathsf{pRST}} \cup \{\mathtt{prst}\}$

[13] In practice, 0-RTT replay attacks can be mounted to *different* servers with the same public-secret key pair. However, 0-RTT key exchange message(s) replayed to other servers with different public-secret key pairs will be rejected.

[14] Note that $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ queries are not needed because msACCE-pauth does not consider data privacy explicitly.

is allowed to be queried.) If $m = \texttt{rst}$, this query asks π_P^i to output its reset packet (if any) and returns it.

This query allows the adversary to specify any associated data and any message in a secure channel phase, then get the packet output by the specified session oracle. The adversary can also specify a session oracle to get the packet resulting from encrypting the session oracle's pre-reset message (which the adversary does not know) or get its reset packet.

• $\mathsf{Deliver}(\pi_P^i, pkt)$, for $P \in \mathcal{P}, i \in [N], pkt \in \{0,1\}^*$.
This query returns $\perp$ if π_P^i is not in a secure channel phase. Otherwise, it delivers pkt to π_P^i and returns its response.

This query allows the adversary to deliver any packet to a specified session oracle and get its response in a secure channel phase.

Advantage Measures. An adversary A against a msACCE protocol Π in msACCE-pauth has the following associated advantage measures.

• *IP-Spoofing Prevention.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{ipsp}}(A)$ as the probability that there exist a client oracle π_C^i and a server oracle π_S^j such that the following holds:

1. π_S^j has set its first stage key right after a $\mathsf{Send}(\pi_S^j, (\mathrm{IP}_C, \mathrm{IP}_S, \cdot, \cdot))$ query;
2. S was not corrupted before π_S^j set its first stage key;
3. The only allowed queries concerning both C and S in the time period associated with π_S^j are:
   - $\mathsf{Connprivate}(\pi_C^x, \pi_S^y, \cdot)$ for any $x, y \in [N]$, and
   - $\mathsf{Send}(\pi_S^y, (\mathrm{IP}_C, \mathrm{IP}_S, \cdot, \cdot))$ for any $y \in [N]$, where $(\mathrm{IP}_C, \mathrm{IP}_S, \cdot, \cdot)$ is the last packet received by π_S^y right before it sets its first stage key.

The above captures the attacks in which the adversary fools a server into accepting a spurious connection request seemingly from an impersonated client, without observing any previous communication between the client and server in the same time period.

• *KE Header Integrity.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{int\text{-}keh}}(A)$ as the probability that there exist a client oracle π_C^i and a server oracle π_S^j such that the following holds:

1. π_C^i has set its session key and has a matching conversation with π_S^j;
2. S was not corrupted before π_C^i set its session key;
3. No interim keys of π_C^i or its peer(s) were revealed;
4. In a key exchange phase before π_C^i set its session key, π_C^i (resp. π_S^j) accepted a packet with a new header that was not output by π_S^j (resp. π_C^i).

The above captures the attacks in which the adversary modifies the protocol header of a key exchange packet of the communicating parties without affecting the client setting its session key. In the above definition, we assume that a client sets its session key *immediately* after sending its last key exchange packet(s) (if any). Then, a forged packet that leads to a successful attack cannot be any of these last packet(s), which have not yet been sent to the server. The same assumption is made for KE Payload Integrity defined below.

• *KE Payload Integrity.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{int\text{-}kep}}(A)$ as the probability that there exist a client oracle π_C^i and a server oracle π_S^j such that the same 1∼3 conditions as in the above KE Header Integrity notion and the following holds:

4. In a key exchange phase before π_C^i set its session key, π_C^i (resp. π_S^j) accepted a packet with a new payload that was not output by π_S^j (resp. π_C^i).

The above captures the attacks in which the adversary modifies the payload of a key exchange packet of the communicating parties without affecting the client setting its session key.

• *SC Header Integrity.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{int\text{-}h}}(A)$ as the probability that A outputs (P, i, d) such that the following holds:

1. If $P = S \in \mathcal{S}$, π_S^i has a matching conversation with a client oracle π_C^j; if $P = C \in \mathcal{C}$, denote S as π_C^i's target server;
2. S was not corrupted before π_P^i set its last stage key; If *forward secrecy* is not required for the d-th stage keys, S was not corrupted in the same time period associated with π_P^i;
3. No stage keys of π_P^i or its peer(s) were revealed.
4. In the secure channel phase of the d-th stage, π_P^i accepted a non-reset packet with a new header that was not output by its peer(s) (via Pack queries), or π_P^i accepted a non-reset header-only packet.

The above captures the attacks in which the adversary creates a valid non-reset secure channel packet by forging the protocol header. Note that in the above security notion an invalid header forgery is detected *immediately* after the malicious packet is received and processed, while the detection of invalid packet forgeries in a key exchange phase (e.g., for plaintext packets) can be *delayed* to the point when the client sets its session key, according to the definitions of KE Header and Payload Integrity.

• *Reset Authentication.* We define $\mathbf{Adv}_{\Pi}^{\mathsf{rst\text{-}auth}}(A)$ as the probability that A outputs (P, i, d) such that the same 1∼3 conditions as in the above SC Header Integrity notion hold and the following holds:

4. In the secure channel of the d-th stage, π_P^i accepted a packet output by a $\mathsf{Pack}(\cdot, \cdot, \mathtt{prst})$ query to its peer $\pi_{P'}^j$. Later (in the d-th or a later stage), π_P^i accepted a reset packet but A made no $\mathsf{Pack}(\pi_{P'}^j, \cdot, \mathtt{rst})$ queries.

The above captures the attacks in which the adversary forges a valid reset packet. Note that such attacks are *undetectable* by the accepting party, as opposed to a network attacker that simply drops packets.

We say a msACCE protocol Π achieves a security notion in our msACCE security models if the associated advantage is negligible (in λ) or for any *probabilistic-polynomial-time (PPT) A*.

Remark about msACCE Security Model Completeness and Low-Layer Integrity. Since the payload integrity in secure channels is captured by msACCE-std, together with msACCE-pauth our models *completely* capture the authentication (or integrity) of all packet fields in the transport and application layers. Furthermore, msACCE-pauth captures (network-layer) IP-Spoofing Prevention against weaker off-path attackers (i.e., those can only inject packets without observing the communication), but leaves other integrity attacks on low layers (e.g., network, link, and physical layers) uncovered. Such attacks may affect packet forwarding, node-to-node data transfer, or raw data transmission, which are outside the scope of our work.

Table 1. Security Comparison

	TLS 1.3 +TFO	QUIC +UDP	QUIC[TLS] +UDP [9]
0-RTT Key Forward Secrecy [18]	✗	✗	✗
0-RTT Data Anti-Replay [18]	✗	✗	✗
Server Authentication [9]	✓	✓	✓
Channel Security [9]	✓	✓	✓
IP-Spoofing Prevention	✓	✓	✓
KE Header Integrity	✗	✗	✗
KE Payload Integrity	✓	✗	✗
SC Header Integrity	✗	✓	✓
Reset Authentication	✗	✗	✓

5 Provable Security Analysis

We now analyze and compare the security of TFO+TLS 1.3 and UDP+QUIC, and refer to the full version [9] for the security analysis of UDP+QUIC[TLS]. The security results are summarized in Table 1. As mentioned in the Introduction, by [18] results, no protocol achieves forward secrecy for 0-RTT keys or protects against 0-RTT data replays (which contribute to the first two rows in the table). The third and fourth rows reflect security results in our basic msACCE-std model (see the full version [9] for detailed analyses), which are derived by adapting existing security results [16,18,33,35] to our model. We now move to the detailed msACCE-pauth security analyses and start with TFO+TLS 1.3.

5.1 TLS 1.3 over TFO

We refer to Appendix A.1 for TFO+TLS 1.3's protocol definition. Its session identifier $\texttt{sid}_{\text{TLS}}$ is defined as all key exchange messages from `ClientHello` to `ServerFinished`, excluding TCP headers and IP addresses. The msACCE-pauth security analyses are shown as follows.

IP-Spoofing Prevention. This security of TFO+TLS 1.3 is provided by the TFO component through TCP sequence number randomization and TFO cookies. By modeling the cookie generation function, an AES-128 block cipher, as a PRF $F : \{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^n$, we have the following theorem with the proof in the full version [9]:

Theorem 1. *For any PPT adversary A making at most q* Send *queries, there exists a PPT adversary B such that:*

$$\mathbf{Adv}^{\mathsf{ipsp}}_{TFO+TLS\ 1.3}(A) \leq |\mathcal{S}|\mathbf{Adv}^{\mathsf{prf}}_{F}(B) + \frac{q}{\min\{2^{|\mathsf{sqn}|}, 2^n\}}\ .$$

KE Header Integrity. TFO+TLS 1.3 does not achieve this security notion because TCP headers are never authenticated. We find a new practical attack below, where a PPT adversary A can always get $\mathbf{Adv}^{\mathsf{int\text{-}keh}}_{\text{TFO+TLS 1.3}}(A) = 1$:

TFO Cookie Removal. A can first make $\pi_C^{i'}$ complete a full handshake with $\pi_S^{j'}$ (via $\mathsf{Connect}, \mathsf{Send}$ queries), then query $\mathsf{Resume}(\pi_C^i, \pi_S^j, i')$ $(i > i', j > j')$ to get the output packet $(\mathrm{IP}_C, \mathrm{IP}_S, H, pd)$, which is a SYN packet with a TFO cookie. A then modifies the `opt` field of H to get a new $H' \neq H$ that contains no cookie. The resulting SYN packet will be accepted by π_S^j, which will then respond with a SYN-ACK packet that does not contain a TFO cookie, indicating a fallback to the standard 3-way TCP. As a result, a 1-RTT handshake is needed to complete the connection and any 0-RTT data sent with SYN would be retransmitted. This eliminates the entire benefit of TFO without being detected, resulting in reduced performance and increased handshake latency. A similar attack is possible by removing the TFO cookie in a server's SYN-ACK packet.

Interestingly, clients are supposed to cache negative TFO responses and avoid sending TFO connections again for a lengthy period of time. This is because the most likely explanation for this behavior is that the server does not support TFO, but only standard TCP [10]. As a result, performing this attack for a single connection prevents TFO from being used with this server for a lengthy time period (i.e., days or weeks).

KE Payload Integrity. TFO+TLS 1.3 is secure in this regard simply because $\mathtt{sid}_{\mathrm{TLS}}$ consists of the payloads of all key exchange packets exchanged between the communicating parties before the client set its session key. That is, for any client oracle that has a matching conversation with any server oracle, by definition they observe the same $\mathtt{sid}_{\mathrm{TLS}}$ and hence no key exchange packet payload can be modified, i.e., $\mathbf{Adv}^{\mathsf{int\text{-}kep}}_{\mathrm{TFO+TLS\ 1.3}}(A) = 0$ for any PPT adversary A.

SC Header Integrity. TFO+TLS 1.3 does not achieve this security again because of the unauthenticated TCP headers. A PPT adversary A can get $\mathbf{Adv}^{\mathsf{int\text{-}h}}_{\mathrm{TFO+TLS\ 1.3}}(A) = 1$ by either modifying the TCP header of an encrypted packet (e.g., reducing the `window` value) or by forging a header-only packet (e.g., removing the payload of an encrypted packet and changing its `ack` value). Such packets are valid and will be accepted by the receiving session oracle.

The above fact exposes the adversary's ability to arbitrarily modify or even entirely forge the information in the TCP header, which is being relied on to provide reliable delivery, in-order delivery, flow control, and congestion control for the targeted flow. This leads to a whole host of availability attacks that the networking community has been slowly uncovering via manual investigation over the last 30 years [46,27,2,8,31,30,25,41,7,21,40,36,48,26]. Some of the practical attacks are described in the full version [9].

Reset Authentication. TFO+TLS 1.3 is insecure in this sense because its reset packet, TCP Reset, is an unauthenticated header-only packet. This leads to a practical attack below, where a PPT adversary A always gets $\mathbf{Adv}^{\mathsf{rst\text{-}auth}}_{\mathrm{TFO+TLS\ 1.3}}(A) = 1$:

TCP Reset Attack. A can first make two session oracles complete a handshake using $\mathsf{Connect}, \mathsf{Send}$ queries, then use $\mathsf{Pack}, \mathsf{Deliver}$ queries to let them exchange secure channel packets. By observing these packet headers, A can easily forge a valid reset packet by setting its RST bit to 1 and the remaining header fields

to reasonable values. This attack will cause TCP to tear down the connection immediately without waiting for all data to be delivered.

Note that even an off-path adversary who can only inject packets into the communication channel may be able to accomplish this attack. The injected TCP reset packet needs to be within the receive window for the client or server, but [48] demonstrated that a surprisingly small number of packets is needed to achieve this, thanks to the large receive windows typically used by implementations.

5.2 QUIC over UDP

We refer to Appendix A.2 for UDP+QUIC's protocol definition. Its session identifier $\texttt{sid}_{\mathrm{QUIC}}$ is defined as the `ClientHello` payload and `ServerHello`, excluding IP addresses. The msACCE-pauth security analyses are shown as follows.

IP-Spoofing Prevention. In [35], QUIC has been proven secure against IP spoofing based on the AEAD security. Their IP-spoofing security notion is the same as our IP-Spoofing Prevention notion for UDP+QUIC except that ours additionally captures attacks in full sessions. However, since source-address tokens are validated in both full and resumption sessions, their results can be trivially adapted to show that UDP+QUIC achieves IP-Spoofing Prevention.

KE Header and Payload Integrity. UDP+QUIC does not achieve these security notions because its first-round key exchange messages, i.e., `InchoateClientHello` and `ServerReject`, and any invalid `ClientHello` are not fully authenticated. Interestingly, a variety of existing attacks on QUIC's availability discovered in [35] are all examples of key exchange packet manipulations (e.g., the server config replay attack, connection ID manipulation attack, etc.), but these attacks cause connection failure and hence are easy to detect. However, successful attacks breaking KE Header or Payload Integrity will be harder (if not impossible) to detect.

For KE Header Integrity, we do not find any harmful attacks but theoretical attacks exist. For instance, a PPT adversary A can get $\mathbf{Adv}^{\mathsf{int\text{-}keh}}_{\mathrm{UDP+QUIC}}(A) = 1$ as follows. A can first query $\mathsf{Connect}(\pi^i_C, \pi^j_S)$ to get the output packet $(\mathrm{IP}_C, \mathrm{IP}_S, H, pd)$, then modify the `flag` and `sqn` fields of H to get a new header $H' \neq H$ that only changes `sqn`'s length but not its value. The resulting packet will be accepted by π^j_S. This attack has no practical impact on UDP+QUIC but it successfully modifies the protocol header without being detected.

For KE Payload Integrity, we find a new practical attack described below where a PPT adversary A can get $\mathbf{Adv}^{\mathsf{int\text{-}kep}}_{\mathrm{UDP+QUIC}}(A) \approx 1$:

ServerReject Triggering. A can first let $\pi^{i'}_C$ complete a full handshake with $\pi^{j'}_S$ with $\mathsf{Connect}, \mathsf{Send}$ queries, then query $\mathsf{Resume}(\pi^i_C, \pi^j_S, i')$ $(i > i', j > j')$ to get the output `ClientHello` packet. A then modifies its payload by replacing the source-address token `stk` with a random value, which with high probability is invalid. Sending this modified packet to π^j_S will trigger a `ServerReject` packet containing a new valid `stk`. This as a result downgrades the original 0-RTT resumption connection to a full 1-RTT connection, which causes increased latency

and results in the retransmission of any 0-RTT data. Note that this attack is hard to detect because π_C^i may think its original $\mathtt{stk}'$ has expired (although this does not happen frequently).

SC Header Integrity. UDP+QUIC is secure in this regard because it does not allow header-only packets to be sent in the secure channel phases and the *entire* protocol header is taken as the associated data authenticated by the underlying AEAD scheme. Therefore, UDP+QUIC's SC Header Integrity can be reduced to its level-1 Channel Security. Formally, for any PPT adversary A there exists a PPT adversary B such that $\mathbf{Adv}^{\mathsf{int\text{-}h}}_{\mathrm{UDP+QUIC}}(A) \leq 2\mathbf{Adv}^{\mathsf{cs\text{-}1}}_{\mathrm{UDP+QUIC}}(B)$, where the constant 2 is due to advantage definition differences between creating a valid forgery and guessing a correct bit.

Reset Authentication. UDP+QUIC does not achieve this security notion because, similar to TCP Reset, its reset packet `PublicReset` is not authenticated either. In the following availability attack, a PPT adversary A can always get $\mathbf{Adv}^{\mathsf{rst\text{-}auth}}_{\mathrm{UDP+QUIC}}(A) = 1$:
PublicReset Attack. A can first make two session oracles complete a handshake using $\mathsf{Connect}, \mathsf{Send}$ queries, then use $\mathsf{Pack}, \mathsf{Deliver}$ queries to let them exchange secure channel packets. By observing these packet headers, A can easily forge a valid (plaintext) reset packet by setting its PUBLIC_FLAG_RESET bit to 1 and the remaining packet fields to reasonable values (which is easy because it simply contains the connection ID `cid`, the sequence number of the rejected packet, and a nonce to prevent replay). This attack will cause similar effects as described in the TCP Reset attack. Note that this vulnerability is fixed in QUIC[TLS] (see the full version [9]).

6 Conclusion

Our work is the first to provide a thorough, formal, and fine-grained security comparison of the most efficient secure channel establishment protocols on the market today. By including packet-level attacks in our analysis, our results shed light on how the reliability, flow control, and congestion control of TFO+TLS 1.3, UDP+QUIC, and UDP+QUIC[TLS] compare besides their basic security, in adversarial settings.

We found that availability functionalities provided by transport-layer protocols like TCP can be easily compromised without packet-level authentication, which may undermine the performance of their supporting application-layer protocols. To protect against availability attacks, new protocols should better implement and authenticate their own transport functionalities like QUIC does. Besides, the key exchange packet integrity should also be scrutinized to avoid serious undetectable availability attacks.

Acknowledgments

We thank the anonymous reviewers for their comments. This paper is based upon work supported by the National Science Foundation under Grant No. 1422794.

References

1. HTTPS encryption on the web - Google transparency report (2018), `https://transparencyreport.google.com/https/overview`
2. Abramov, R., Herzberg, A.: TCP ack storm DoS attacks. In: IFIP International Information Security Conference. pp. 29–40 (2011)
3. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Annual International Cryptology Conference. pp. 232–249. Springer (1993)
4. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the tls 1.3 standard candidate. In: Security and Privacy (SP). pp. 483–502. IEEE (2017)
5. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zanella-Béguelin, S.: Proving the TLS handshake secure (as it is). In: Proceedings of CRYPTO (2014)
6. Brendel, J., Fischlin, M., Günther, F.: Breakdown resilience of key exchange protocols and the cases of newhope and tls 1.3. Cryptology ePrint Archive, Report 2017/1252 (2017)
7. Cao, Y., Qian, Z., Wang, Z., Dao, T., Krishnamurthy, S.V., Marvel, L.M.: Off-path TCP exploits: Global rate limit considered dangerous. In: USENIX Security Symposium (2016)
8. Centre for the Protection of National Infrastructure: Security assessment of the transmission control protocol. Tech. Rep. CPNI Technical Note 3/2009, Centre for the Protection of National Infrastructure (2009)
9. Chen, S., Jero, S., Jagielski, M., Boldyreva, A., Nita-Rotaru, C.: Secure communication channel establishment: Tls 1.3 (over tcp fast open) vs. quic. Cryptology ePrint Archive, Report 2019/433 (2019), `https://eprint.iacr.org/2019/433`
10. Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A.: TCP Fast Open. RFC 7413 (Experimental) (Dec 2014)
11. Cremers, C., Horvat, M., Scott, S., v. Merwe, T.: Automated analysis and verification of tls 1.3: 0-rtt, resumption and delayed authentication. In: 2016 IEEE Symposium on Security and Privacy (SP). vol. 00, pp. 470–485 (2016). https://doi.org/10.1109/SP.2016.35
12. Cremers, C., Horvat, M., Hoyland, J., Scott, S., van der Merwe, T.: A comprehensive symbolic analysis of tls 1.3. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1773–1788. ACM (2017)
13. Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Protzenko, J., Rastogi, A., Swamy, N., Béguelin, S.Z., Bhargavan, K., Pan, J., Zinzindohoue, J.K.: Implementing and proving the TLS 1.3 record layer. In: 2017 IEEE Symposium on Security and Privacy, SP 2017. pp. 463–482. IEEE Computer Society (2017)
14. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the tls 1.3 handshake protocol candidates. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 1197–1210. CCS '15, ACM, New York, NY, USA (2015)
15. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the tls 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081 (2016), `https://eprint.iacr.org/2016/081`
16. Dowling, B.J.: Provable security of internet protocols. Ph.D. thesis, Queensland University of Technology (2017)
17. Fischlin, M., Günther, F.: Multi-stage key exchange and the case of google's quic protocol. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1193–1204. ACM (2014)

18. Fischlin, M., Günther, F.: Replay attacks on zero round-trip time: The case of the tls 1.3 handshake candidates. In: Security and Privacy (EuroS&P), 2017 IEEE European Symposium on. pp. 60–75. IEEE (2017)
19. Fischlin, M., Günther, F., Marson, G.A., Paterson, K.G.: Data is a stream: Security of stream-based channels. In: Annual Cryptology Conference. pp. 545–564. Springer (2015)
20. Gebhart, G.: Tipping the scales on https: 2017 in review (December 2017), https://www.eff.org/deeplinks/2017/12/tipping-scales-https
21. Gilad, Y., Herzberg, A.: Off-path attacking the web. In: WOOT. pp. 41–52 (2012)
22. IP Latency Statistics — Verizon Enterprise Solutions: Verizon Enterprise Solutions (2018), http://www.verizonenterprise.com/about/network/latency/
23. Iyengar, J., Thomson, M.: Quic: A udp-based multiplexed and secure transport (January 2019), https://quicwg.org/base-drafts/draft-ietf-quic-transport.html
24. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of tls-dhe in the standard model. In: Advances in Cryptology–CRYPTO 2012, pp. 273–293. Springer (2012)
25. Jero, S., Lee, H., Nita-Rotaru, C.: Leveraging State Information for Automated Attack Discovery in Transport Protocol Implementations. In: IEEE/IFIP International Conference on Dependable Systems and Networks (2015)
26. Jero, S., Hoque, E., Choffnes, D., Mislove, A., Nita-Rotaru, C.: Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach. In: Network and Distributed Systems Security Symposium (NDSS) (2018)
27. Joncheray, L.: A simple active attack against TCP. In: USENIX Security Symposium (1995)
28. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the tls protocol: A systematic analysis. In: Advances in Cryptology–CRYPTO 2013, pp. 429–448. Springer (2013)
29. Krawczyk, H., Wee, H.: The optls protocol and tls 1.3. In: Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. pp. 81–96. IEEE (2016)
30. Kumar, V.A., Jayalekshmy, P.S., Patra, G.K., Thangavelu, R.P.: On remote exploitation of TCP sender for low-rate flooding denial-of-service attack. IEEE Communications Letters **13**(1), 46–48 (2009)
31. Kuzmanovic, A., Knightly, E.: Low-rate TCP-targeted denial of service attacks and counter strategies. IEEE/ACM Transactions on Networking **14**(4), 683–696 (2006)
32. Langley, A., Chang, W.: Quic crypto (2016), https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit
33. Li, X., Xu, J., Zhang, Z., Feng, D., Hu, H.: Multiple handshakes security of tls 1.3 candidates. In: Security and Privacy (SP), 2016 IEEE Symposium on. pp. 486–505. IEEE (2016)
34. Linden, G.: Make data useful (2006), https://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt
35. Lychev, R., Jero, S., Boldyreva, A., Nita-Rotaru, C.: How secure and quick is quic? provable security and performance analyses. In: Security and Privacy (SP), 2015 IEEE Symposium on. pp. 214–231. IEEE (2015)
36. Morris, R.: A weakness in the 4.2 BSD unix TCP/IP software. Tech. rep., AT&T Bell Leboratories (1985)
37. Patton, C., Shrimpton, T.: Partially specified channels: The TLS 1.3 record layer without elision. In: ACM SIGSAC Conference on Computer and Communications Security. ACM (2018)

38. Postel, J.: User datagram protocol. RFC 768 (Standard) (1980)
39. Postel, J.: Transmission control protocol. RFC 793 (Standard) (1981)
40. Qian, Z., Mao, Z.M.: Off-path TCP sequence number inference attack - how firewall middleboxes reduce security. In: IEEE Symposium on Security and Privacy. pp. 347–361 (2012)
41. Qian, Z., Mao, Z.M., Xie, Y.: Collaborative TCP sequence number inference attack: how to crack sequence number under a second. In: ACM Conference on Computer and Communications Security (2012)
42. Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., Raghavan, B.: Tcp fast open. In: Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies. p. 21. ACM (2011)
43. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018)
44. Rogaway, P.: Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM conference on Computer and communications security. pp. 98–107. ACM (2002)
45. Roskind, J.: Quic(quick udp internet connections): Multiplexed stream transport over udp. Technical report, Google (2013)
46. Savage, S., Cardwell, N., Wetherall, D., Anderson, T.: TCP congestion control with a misbehaving receiver. ACM SIGCOMM Computer Communication Review **29**(5) (1999)
47. Thomson, M., Turner, S.: Using transport layer security (tls) to secure quic (January 2019), `https://quicwg.org/base-drafts/draft-ietf-quic-tls.html`
48. Watson, P.: Slipping in the window: TCP reset attacks. Tech. rep., CanSecWest (2004), `http://bandwidthco.com/whitepapers/netforensics/tcpip/TCPResetAttacks.pdf`

A TFO+TLS 1.3 and UDP+QUIC Protocol Definitions

A.1 TFO+TLS 1.3 Protocol Definition

Referring to the msACCE protocol syntax, a TFO+TLS 1.3 2-RTT full handshake (see Fig. 1) is a 2-stage msACCE protocol in the full mode and a 0-RTT resumption handshake (see Fig. 1) is a 3-stage msACCE protocol in the resumption mode. Note that we focus only on the main components of the handshakes and omit more advanced features such as 0.5-RTT data, client authentication, and post-handshake messages (except `NewSessionTicket`). In a full handshake, the initial keys are set after sending or receiving `ServerHello` and the final keys (i.e., session keys) are set after sending or receiving `ClientFinished` (but only handshake messages up to `ServerFinished` are used for final key generation). In a 0-RTT resumption handshake, the parties set 0-RTT keys to encrypt or decrypt 0-RTT data, after sending or receiving `ClientHello`.

According to the TFO and TLS 1.3 specifications [10,43], the TFO+TLS 1.3 header contains the TCP header [39]. We ignore some uninteresting header fields such as port numbers and the checksum because modifying them only leads to redirected or dropped packets. Such adversarial capabilities are already considered in the msACCE security models. We thus define the header space $\mathcal{H}$ as containing the following fields: a 32-bit sequence number `sqn`, a 32-bit

acknowledgment number `ack`, a 4-bit data offset `off`, a 6-bit reserved field `resvd`, a 6-bit control bits field `ctrl`, a 16-bit window `window`, a 16-bit urgent pointer `urgp`, a variable-length ($\leq$ 320-bit) padded options `opt`. For encrypted packets, $\mathcal{H}$ additionally contains the TLS 1.3 record header fields: an 8-bit type `type`, a 16-bit version `ver`, and a 16-bit length `len`. We further define reset packets as those with the RST bit (i.e., the 4-th bit of `ctrl`) set to 1. Note that $\mathsf{scfg_gen}$ is undefined.

TLS 1.3 enforces different content types for encrypted key exchange and secure channel messages. For simplicity, we define $\mathcal{M}_{\mathsf{KE}}$ and $\mathcal{M}_{\mathsf{SC}}$ as consisting of bit strings differing in their first bits. $\mathcal{M}_{\mathsf{pRST}} = \varnothing$. We refer to the full version [9] for the remaining TFO details and to [18,6] for the detailed descriptions of TLS 1.3 handshake messages and key generations in earlier TLS 1.3 drafts as well as [43] for the latest updates.

A.2 UDP+QUIC Protocol Definition

Referring to the msACCE protocol syntax, an UDP+QUIC 1-RTT full handshake (see Fig. 2) is a 2-stage msACCE protocol in the full mode and a 0-RTT resumption handshake (see Fig. 2) is a 2-stage msACCE protocol in the resumption mode. The initial keys are set after sending or receiving `ClientHello` and the final keys (i.e., session keys) are set after sending or receiving `ServerHello`.

According to the UDP and QUIC specifications [45,38,32], the UDP+QUIC header contains the UDP header [38] and the QUIC header (described below). As with the TCP header, we ignore the port numbers and checksum in the UDP header. Similarly, we also ignore the UDP length field because it only affects the length of the QUIC header and payload. We thus can completely omit the UDP header and define the header space $\mathcal{H}$ as containing the following fields: an 8-bit public flag `flag`, a 64-bit connection ID `cid`, a variable-length ($\leq$ 48-bit) sequence number `sqn`, and other optional fields. We further define reset packets as those with the PUBLIC_FLAG_RESET bit (i.e., the 7-th bit of `flag`) set to 1. A reset packet header only contains `flag` and `cid`.

As with TLS 1.3, we define $\mathcal{M}_{\mathsf{KE}}$ and $\mathcal{M}_{\mathsf{SC}}$ as consisting of bit strings differing in their first bits. $\mathcal{M}_{\mathsf{pRST}} = \varnothing$. We refer to [35] for the detailed descriptions of $\mathsf{scfg_gen}$ and QUIC handshake messages and key generations.