Authenticated Handshakes

QUIPS - NDSS 2020

February 23, 2020

https://tools.ietf.org/html/draft-kazuho-quic-authenticated-handshake-00

Overview (HIDE ME)

QUIC handshake and encryption keys, including properties

Obfuscation keys

ESNI (ECHO) overview in TLS

Abbreviated design

Authentication in QUIC

A Normal Handshake

```
Client                                                  Server
    ClientHello
   (0-RTT Application Data)  -------->
                                                   ServerHello
                                         {EncryptedExtensions}
                                                    {Finished}
                             <--------      [Application Data]
   {Finished}                -------->

   [Application Data]        <------->      [Application Data]

    () Indicates messages protected by Early Data (0-RTT) Keys
    {} Indicates messages protected using Handshake Keys
    [] Indicates messages protected using Application Data (1-RTT) Keys
```

https://quicwg.org/base-drafts/draft-ietf-quic-tls.html

Packet Protection Keys

Data is protected using a number of encryption levels:

- **Initial Keys**
- Early Data (0-RTT) Keys
- Handshake Keys
- Application Data (1-RTT) Keys

Initial Keys

Publicly derivable using “cleartext” information

```
initial_salt = 0xc3eef712c72ebb5a11a7d2432bb46365bef9f502
initial_secret = HKDF-Extract(initial_salt, client_dst_connection_id)
client_initial_secret = HKDF-Expand-Label(initial_secret, "client in", "", Hash.length)
server_initial_secret = HKDF-Expand-Label(initial_secret, "server in", "", Hash.length)
```

An on-path adversary can derive the same keys, modify the payload, and insert a new DCID

Initial Authentication

ClientHello and ServerHello are encrypted (without authentication) under Initial packet keys

What can go wrong?

- Attacker causes connection failure due to endpoint decryption failures or mismatched transcripts
- Attacker can tamper with Initial packets arbitrarily

How can we protect the entire handshake from modification?

“Simple” Mitigation

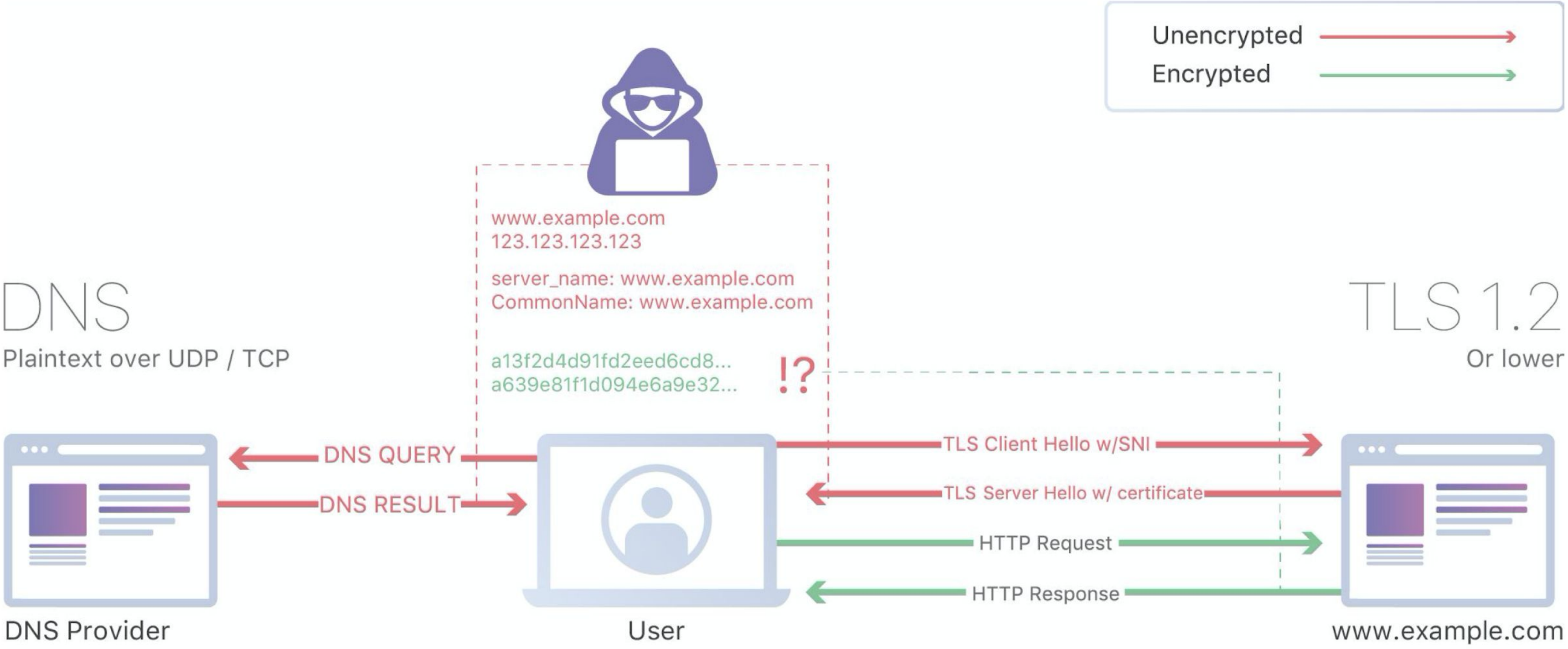
Embed a secret in the ClientInitial and use it to authenticate the packet (and all that follow)

Challenges:

1. Where does this shared secret come from, and how does it fit in ClientInitial?
2. How do we deal with Version Negotiation and Retry packets?

… a brief detour to TLS

TLS 1.2 Handshake Privacy



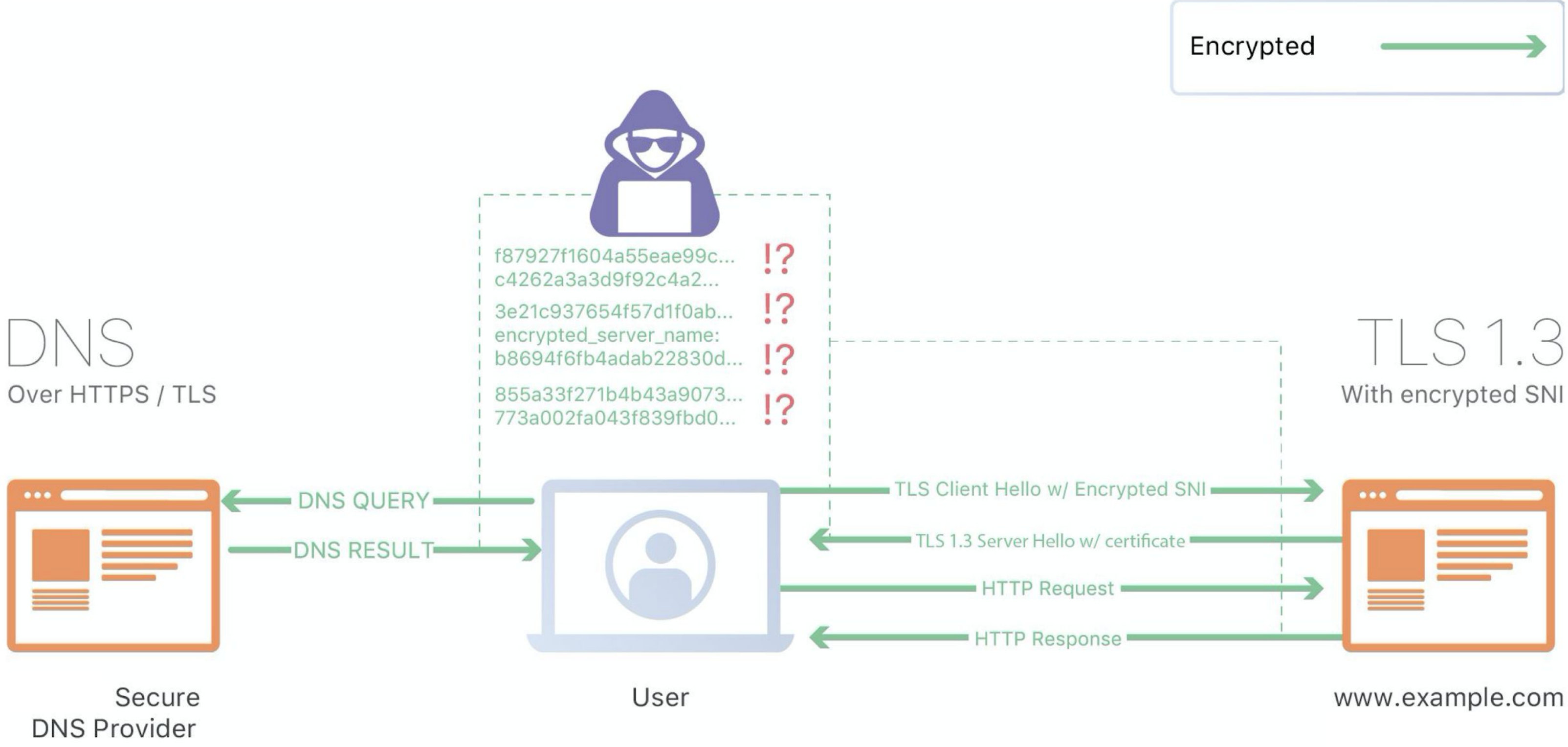
Encrypted Client HellO (ECHO)

Primary goal: *Encrypt* as much of the TLS ClientHello as possible

Requirements

- Mitigate replay attacks
- Avoid widely deployed shared secrets
- Prevent SNI-based DoS attacks
- Do not stick out
- Forward secrecy
- Split server support
- …

ECHO Flow in TLS 1.3



ECHO Mechanics

Sensitive ClientHello contents protected with authenticated public key encryption (HPKE)

- Servers publish *static HPKE key share* in DNS
- Clients *encrypt private ClientHello* using HPKE, and send ciphertext in a *public ClientHello*

Active attacks mitigated by transcript “alterations”*

HPKE-derived shared secrets ***exportable*** to TLS

* This is an active research problem. The design is subject to change!

… back to QUIC

Authenticated Handshake

Challenge: Use shared secret for *Initial packet authentication*

Design questions:

- Where does the shared secret come from?
- Support graceful fallback to unauthenticated handshake if ECHO private keys are lost or rotated?
- Doubly encrypt and authenticate the ClientHello and outer QUIC packet?

Authenticated Handshake

Challenge: Use shared secret for *Initial packet authentication*

Design questions:

- Where does the shared secret come from? **ECHO!**
- Support graceful fallback to unauthenticated handshake if ECHO private keys are lost or rotated? **Yes!**
- Doubly encrypt and authenticate the ClientHello and outer QUIC packet? **Yes-ish!**

ECHO-Based Proposal*

1) Encrypt packet payload with AES-CTR, using “public” Initial keys
2) Derive shared *authentication* secret from ECHO
3) Authenticate the packet header with HMAC

*This is incompatible with QUICv1, so the draft currently uses a **different version** number for these packets

Authenticated Handshake

```
Client                                                 Server
   <ClientHello>
   (0-RTT Application Data)  -------->
                                                  <ServerHello>
                                          {EncryptedExtensions}
                                                     {Finished}
                             <--------      [Application Data]
   {Finished}                -------->

   [Application Data]        <------->      [Application Data]

    <> Indicates messages authenticated by ECHO secrets
    () Indicates messages protected by Early Data (0-RTT) Keys
    {} Indicates messages protected using Handshake Keys
    [] Indicates messages protected using Application Data (1-RTT) Keys
```

https://quicwg.org/base-drafts/draft-ietf-quic-tls.html

Server-Side Processing

Upon ClientInitial*, decrypt and process the TLS ClientHello to derive the ECHO shared secret

Authentication check:

1. Success: Proceed with the connection
2. Failure: Proceed with the connection

Gray area -- should servers proceed as normal, close the connection (hard fail), or do something more robust?

*If the ClientInitial corresponds to an existing connection, check the authentication tag and drop on failure

Server-Side Processing

ClientInitial decrypts and authenticates correctly:

→ Process per normal rules...

ClientInitial decrypts correctly yet fails authentication:

Missing private key? On-path tampering? Both?

→ Process per normal rules...

Server-Side Processing

ClientInitial decrypts and authenticates correctly:

→ Process per normal rules...

ClientInitial decrypts correctly yet fails authentication:

Missing private key? On-path tampering? Both?

→ Process per normal rules...

?

ECHO Authenticated Fallback

In TLS, upon *key mismatch*…

- Server completes the connection using a *public name* and provides fresh ECHO keys to the client
- Client authenticates the server, stores the new keys, and then retries the connection again

Pro: ECHO robustness

Con: Burn a round trip upon key mismatch*

*Decrypt failure yields an alert

Authenticated Handshake (cont'd)

Challenge: Downgrade attacks by attacker-issued Version Negotiation

Design question: How does a client know that the VN packet is legitimate?

Downgrade Prevention

Setup:

- Servers publish supported versions alongside ECHO keys in DNS
- Clients include supported versions inside a TLS extension

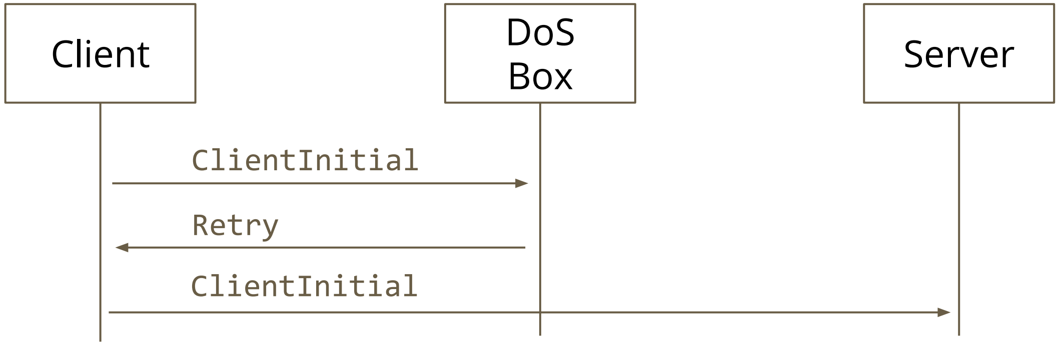
Clients check VN packets for legitimacy against DNS

- Couples servers to DNS packets, which may be a problem for robustness

Authenticated Handshake (cont'd)

Challenge: Retry spoofing

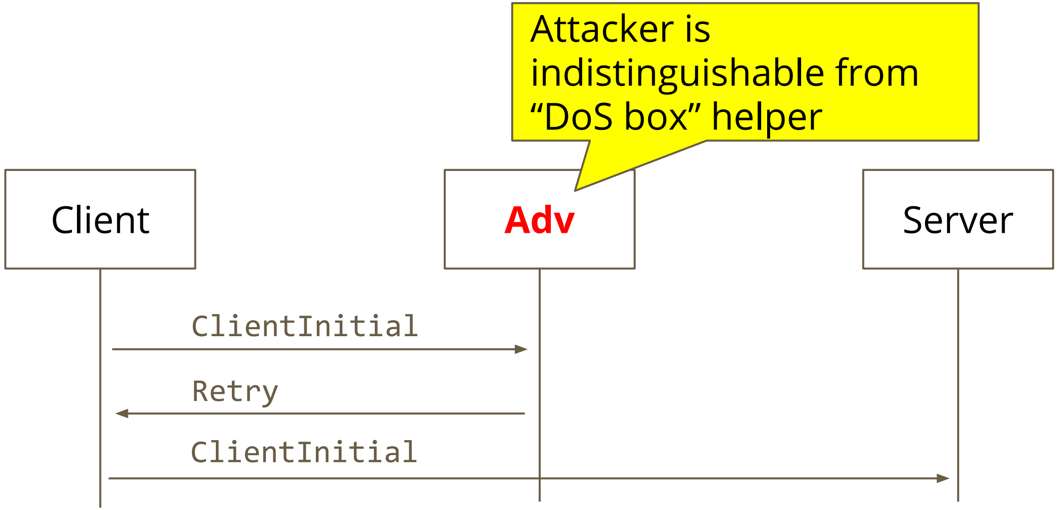
Design consideration: Retry packets likely generated by servers *without* access to ECHO keys



Authenticated Handshake (cont’d)

Challenge: Retry spoofing

Design consideration: Retry packets likely generated by servers *without* access to ECHO keys



Repeat Yourself

Current text:

- Clients calculate new Initial secrets (based on Retry)
- Servers verify (via Transport Parameters) that the second ClientInitial is a continuation of the first

Proposed change: send the same ClientInitial!

- Clients send the *same* CRYPTO contents in the second ClientInitial
- Clients use the original DCID in deriving the Initial secrets

Open Questions

1. Should QUIC servers also fail in the presence of authentication failures?
2. Should we reuse existing TLS AEAD algorithms for Initial packet encryption and authentication?
3. ...?

Next Steps

Update proposal to match upcoming ECHO changes

(Simplified) symbolic analysis to assert desired authentication properties