Encrypted SNI
IETF 102

Eric Rescorla
ekr@rtfm.com

Kazuho Oku
kazuhooku@gmail.com

Nick Sullivan
nick@cloudflare.com

Chris Wood
cawood@apple.com

“Develop a mode that encrypts as much of the handshake as is possible to reduce the amount of observable data to both passive and active attackers.”
-- TLS WG Charter

How did we do?

- Not too bad
  - Most of the server extensions
  - Server certificate
  - Client certificate
- What’s left?
  - Client’s extensions (principally Server Name Indication)

Clients want to conceal the server they are going to

- Why?
  - Surveillance
  - Censorship
- Attack models
  - Active
  - Passive

Sources of Server Identity Leakage

- DNS resolution
- Server Name Indication
- Server certificate
- Server IP address
- Traffic analysis

Sources of Server Identity Leakage

- ~~DNS resolution~~ DPRIVE/DoH
- ~~Server Name Indication~~ This draft
- ~~Server certificate~~ TLS 1.3
- ~~Server IP address~~ CDNs/multi-tenanting*
- Traffic analysis

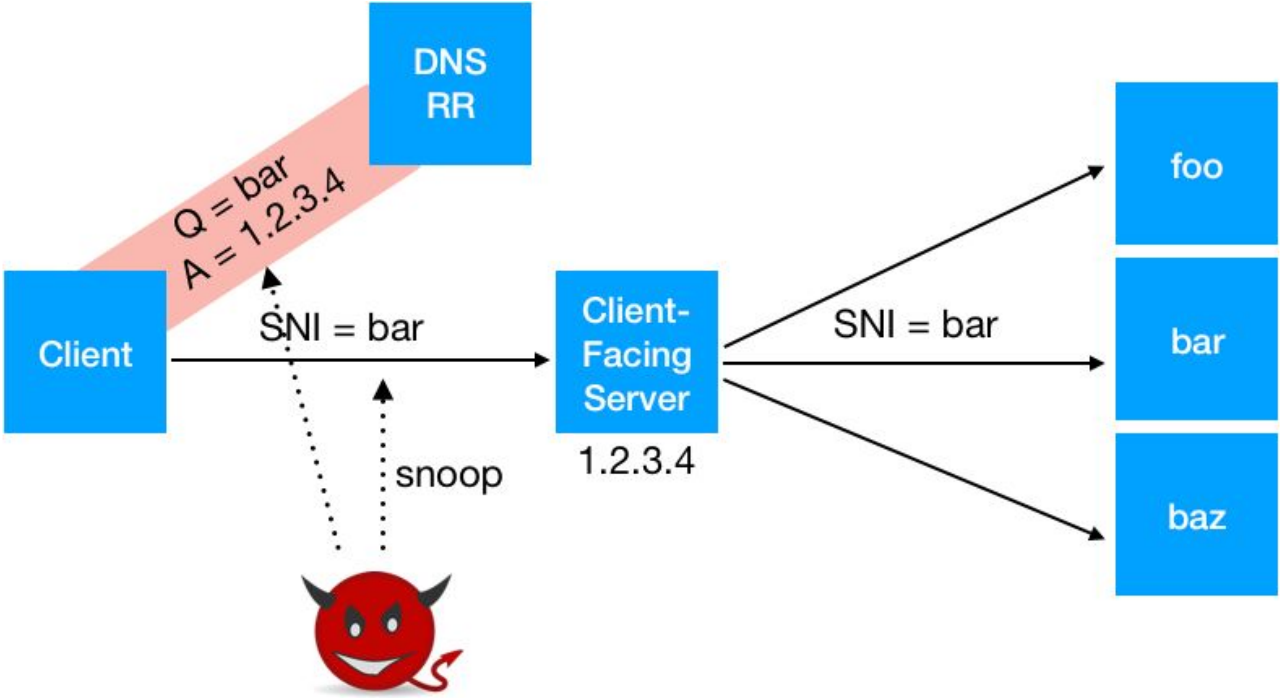
We have spent a *lot* of time on this

- Going back to the start of TLS 1.3
- See also draft-ietf-tls-sni-encryption-03
- Concluded it was really hard
- So what's changed?

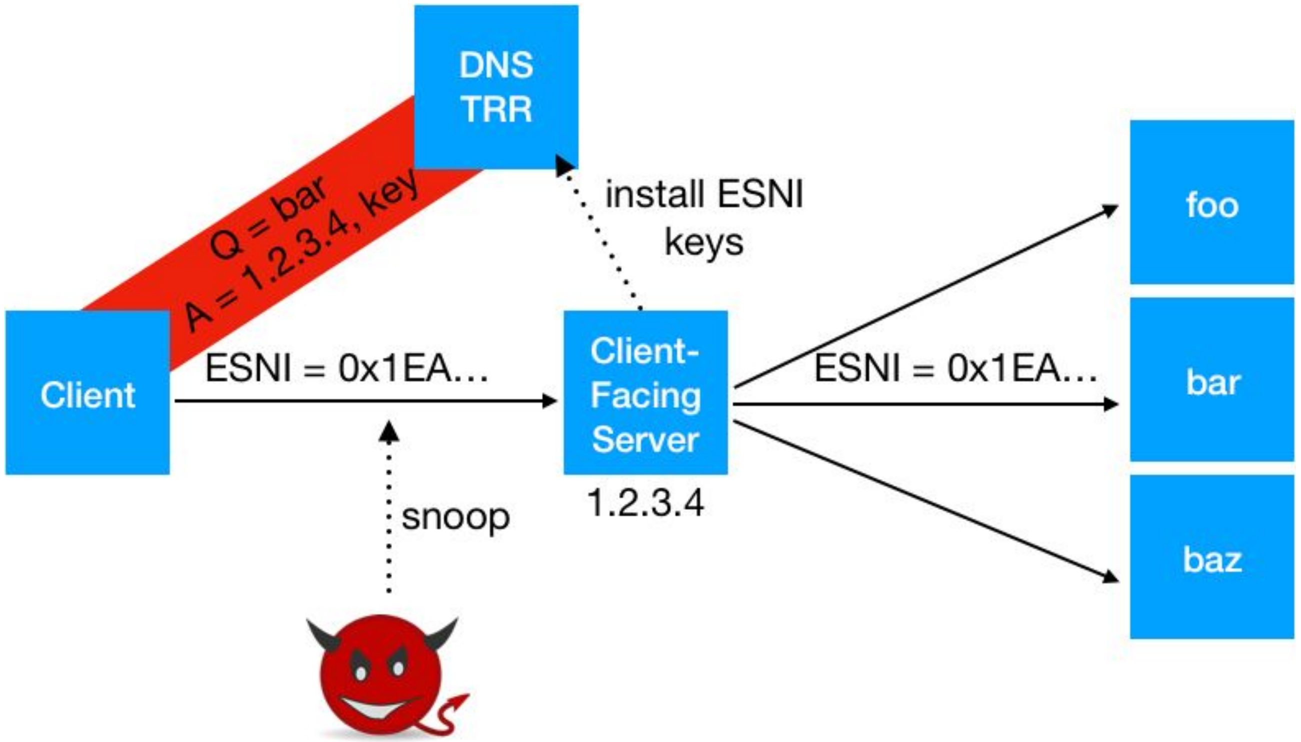
80/20 solution

- Previously we worried about sticking out
  - What if just "sensitive" sites support SNI encryption
  - But what if we could do a mass change?
- A solution that works for CDNs and hosting providers
  - They can mass-reconfigure all their domains
  - Many of them also control DNS for their customers
- This puts everyone behind the same provider in the same anonymity set

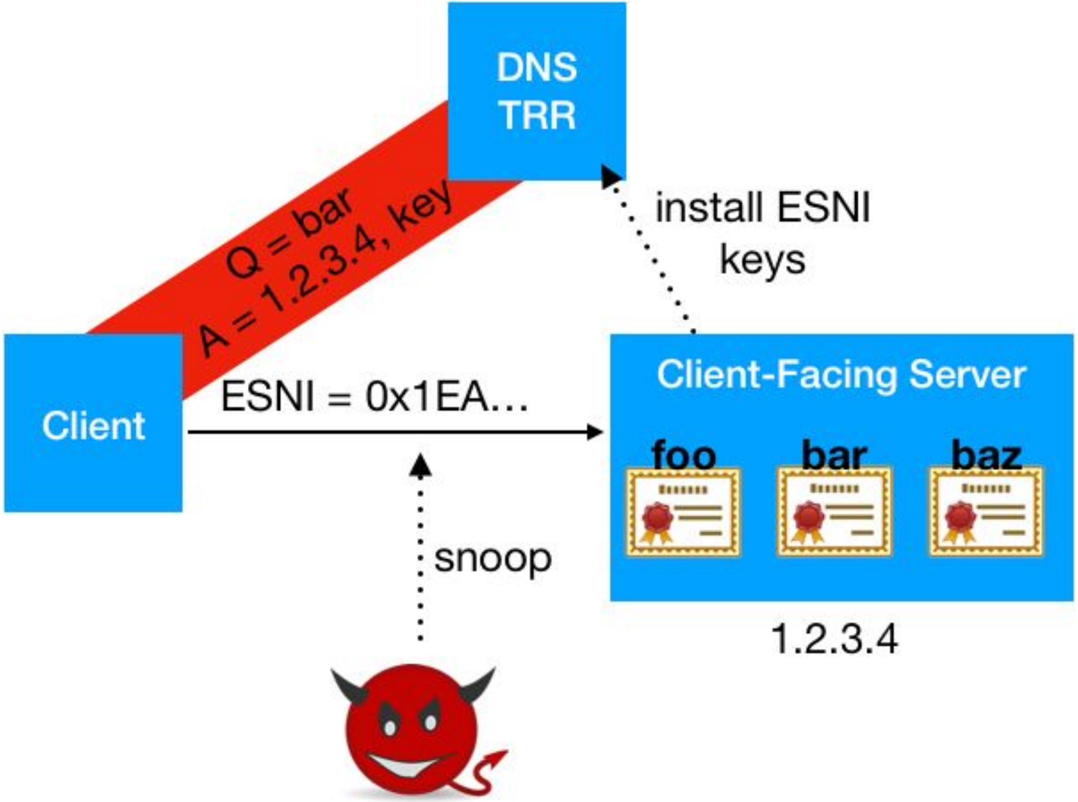
Topologies - Today



Topologies - Split Mode



Topologies - Shared Mode



DNS Pieces

```
struct {
    uint8 checksum[4];
    KeyShareEntry keys<4..2^16-1>;
    CipherSuite cipher_suites<2..2^16-2>;
    uint16 padded_length;
    uint64 not_before;
    uint64 not_after;
    Extension extensions<0..2^16-1>;
} ESNIKeys;
```

TXT record under _esni.example.com

_esni.cloudflare-esni.com. 120 IN TXT
"GpTSIAAkAB0AIICiQKV0aCWs51BnOr19MapPjMeSEmt+0iyd2iu8Q7tIAAITAQEEAAAAAFs/iOgAAAAAW7Yv5wAA"

New TLS Extension

```
struct {
   CipherSuite suite;
   opaque record_digest<0..2^16-1>;
   opaque encrypted_sni<0..2^16-1>;
} EncryptedSNI;
```

- suite: the AEAD algorithm used to encrypt the SNI
- record_digest: the hash of the ESNIKeys record
- encrypted_sni: encryption of the original ServerKeysList structure

Key Derivation

- ESNI-encryption key derived from
  - Client KeyShare from ClientHello
  - A server KeyShare from ESNIKeys structure
- This has some side effects
  - Client chooses and sends one KeyShare for *both* ESNI and the handshake
    - Ciphersuite is still negotiated per usual
  - Client-facing and hidden servers need to share a group
  - Potential for downgrades (more on this later)

Interaction with Middleboxes

- S 9.3 requires middleboxes not to send extensions they don't understand
  - Therefore they will strip the ESNI
  - The server will *likely* respond with a default certificate
  - This will chain to a user-installed trust anchor
  - So we could detect it
- Noncompliant middleboxes create hard failure
  - Not entirely clear how to detect this
  - Some kind of captive portal detection?

How do enterprises disable ESNI?

- Strip ESNIKeys records from DNS? Keep TTLs short?
- Some sort of client policy push
    - You'll want this for DoH as well
- Something else?

Why not just encrypt everything?

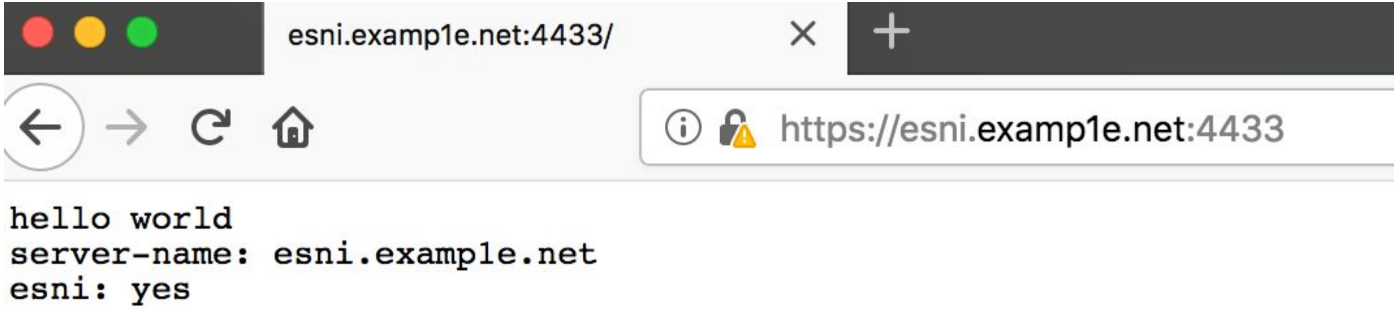
- This interacts poorly with split architecture
  - ESNI permits key separation
- Also means that middleboxes will strip *every* extension
  - Which will certainly cause bustage
- We could later introduce a separate "the rest of the extensions" encrypted extension

This draft is all wrong

- DNS structure
  - Should we remove base64?
  - What about a non-text RR type?
  - Alt-svc instead of _esni record
- TLS
  - Maybe don't reuse key share
    - But need to bind the client KeyShare to ESNI
  - Hand waving: separate ESNIKeyShare/ESNI + KeyShare->ESNI binding
- But it is in the right direction (we think)

Interop Status (mostly not landed)

- Libraries
  - NSS, BoringSSL, PicoTLS
- Browsers
  - Firefox, Safari (experimental, en route)
- Test servers for PicoTLS and BoringSSL (Cloudflare)



WG Interest? Next Steps?