Improving TLSv1.3 privacy
a work-in-progress

Encrypted Server Name Indication
becomes
Encrypted ClientHello

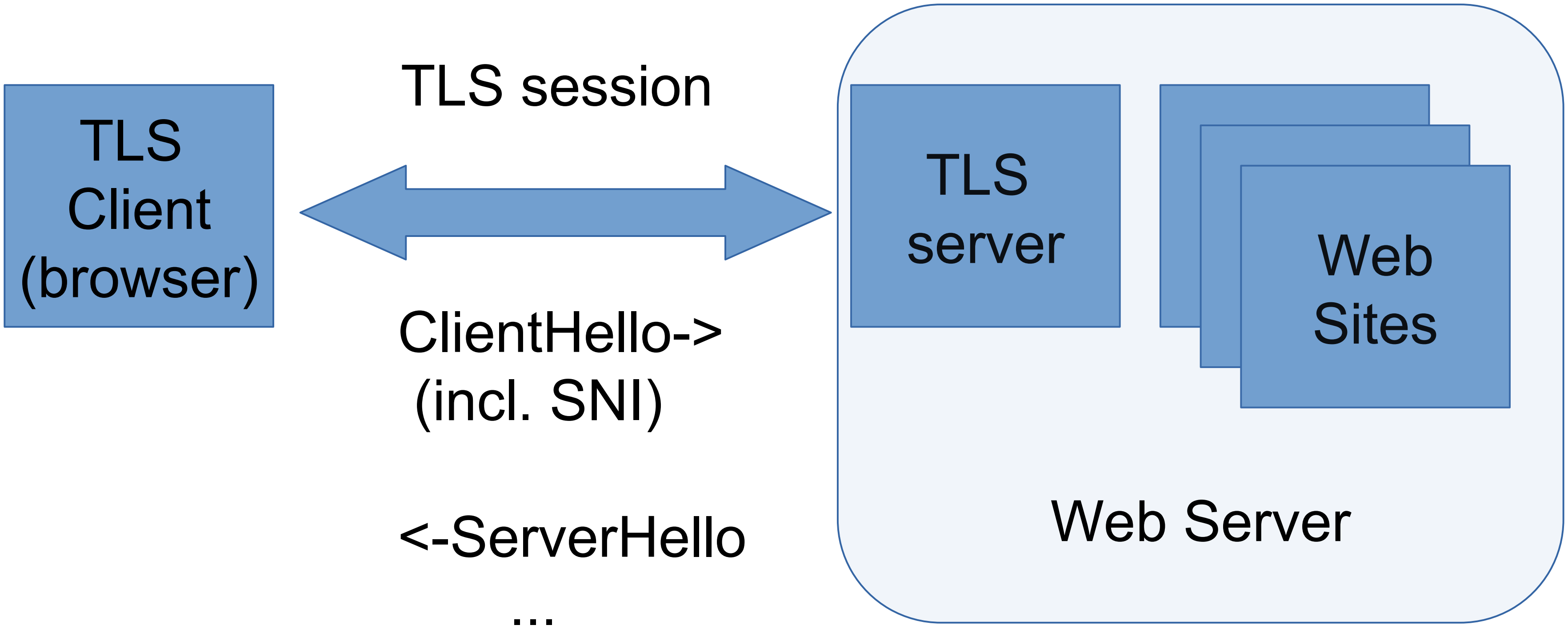
Stephen Farrell
Trinity College Dublin
stephen.farrell@cs.tcd.ie

April 2020

TLS and SNI

- Transport Layer Security (TLS, RFC8446) is the security protocol that secures the web and many other applications – HTTP running over TLS is what makes HTTPS
- One web server instance (e.g. an apache install using more than one VirtualHost) can, and very frequently does, serve multiple web sites
- Each of those may (and is very likely to) use different TLS server key pairs/certificates

TLS for multiple web sites



TLS Client (browser)

TLS session

ClientHello-> (incl. SNI)

<-ServerHello

...

TLS server

Web Sites

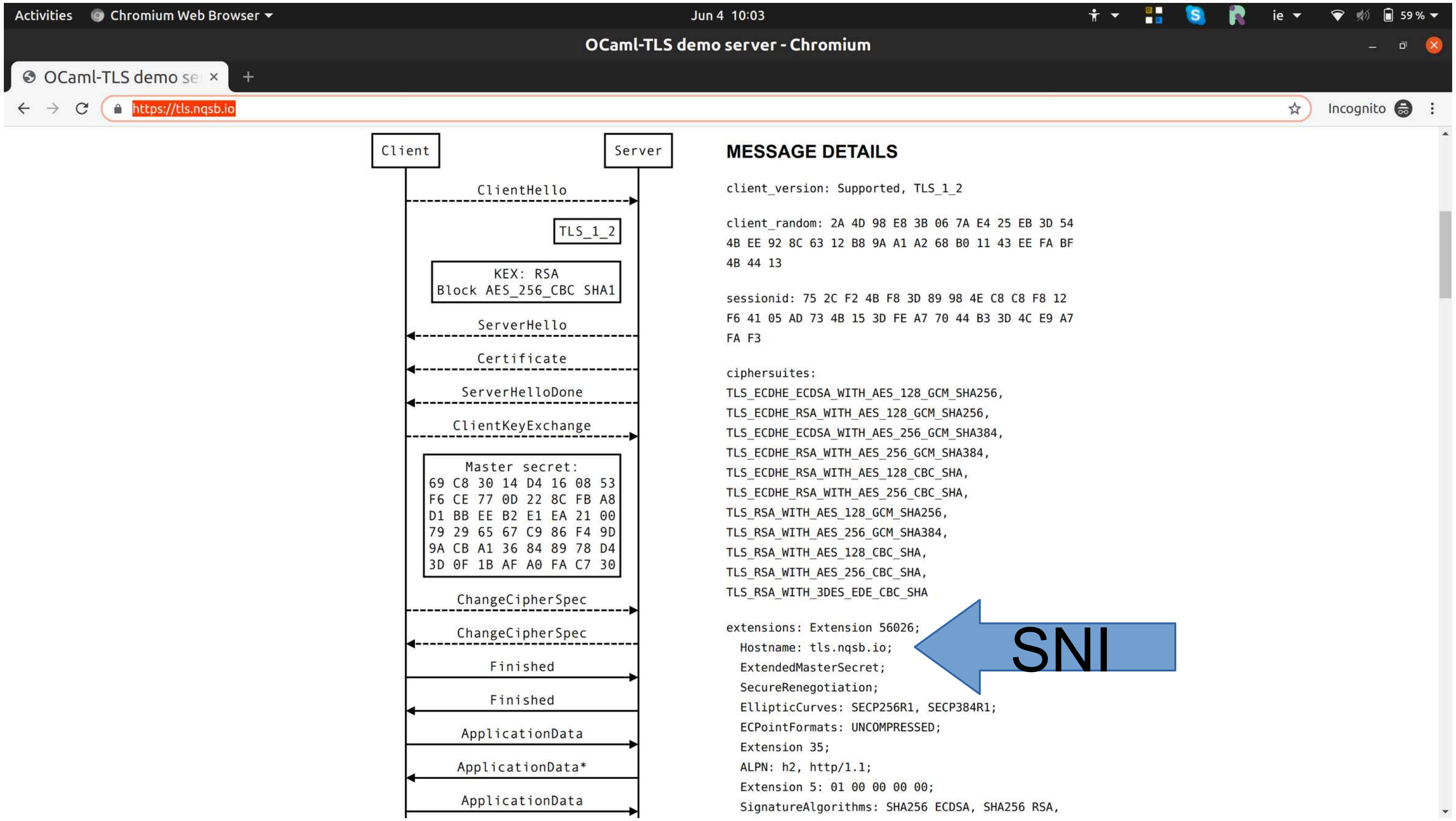
Web Server

TLS and SNI

- One of the first things an HTTPS server needs to do is pick a TLS server key/certificate to use for the TLS session
- The client needs to verify that it’s talking to the correct server via the TLS server certificate, which (for the web) contains the domain name of the web site
- Result: the first TLS message the client sends (the ClientHello) needs to specify the web site (DNS name) for which the TLS session is being established
- That’s done using the Server Name Indication (SNI) extension to the ClientHello (RFC6066)

TLS Handshake

Nice TLS visualisation: https://tls.nqsb.io/



MESSAGE DETAILS

```
client_version: Supported, TLS_1_2

client_random: 2A 4D 98 E8 3B 06 7A E4 25 EB 3D 54
4B EE 92 8C 63 12 B8 9A A1 A2 68 B0 11 43 EE FA BF
4B 44 13

sessionid: 75 2C F2 4B F8 3D 89 98 4E C8 C8 F8 12
F6 41 05 AD 73 4B 15 3D FE A7 70 44 B3 3D 4C E9 A7
FA F3

ciphersuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_128_GCM_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_3DES_EDE_CBC_SHA

extensions: Extension 56026;
  Hostname: tls.nqsb.io;
  ExtendedMasterSecret;
  SecureRenegotiation;
  EllipticCurves: SECP256R1, SECP384R1;
  ECPointFormats: UNCOMPRESSED;
  Extension 35;
  ALPN: h2, http/1.1;
  Extension 5: 01 00 00 00 00;
  SignatureAlgorithms: SHA256 ECDSA, SHA256 RSA,
```

SNI as a leak

- Since the SNI value is sent in the first message, neither party has a key with which to encrypt, so SNI is sent in clear in the first TLS handshake message (the ClientHello)
- When accessing https://bank.example.com/getBalance the “getBalance” part will be encrypted (later, when the full HTTP request with the full URL is sent) but the DNS name (“bank.example.com”) is sent in clear in the SNI extension
- That’s a noticeable leak, especially as the SNI is visible to everyone on the path (my ISP, the site’s ISP, every intermediate router, hosters, governments)
- SNI has also been used for censorship

  https://www.bleepingcomputer.com/news/security/south-korea-is-censoring-the-internet-by-snooping-on-sni-traffic/

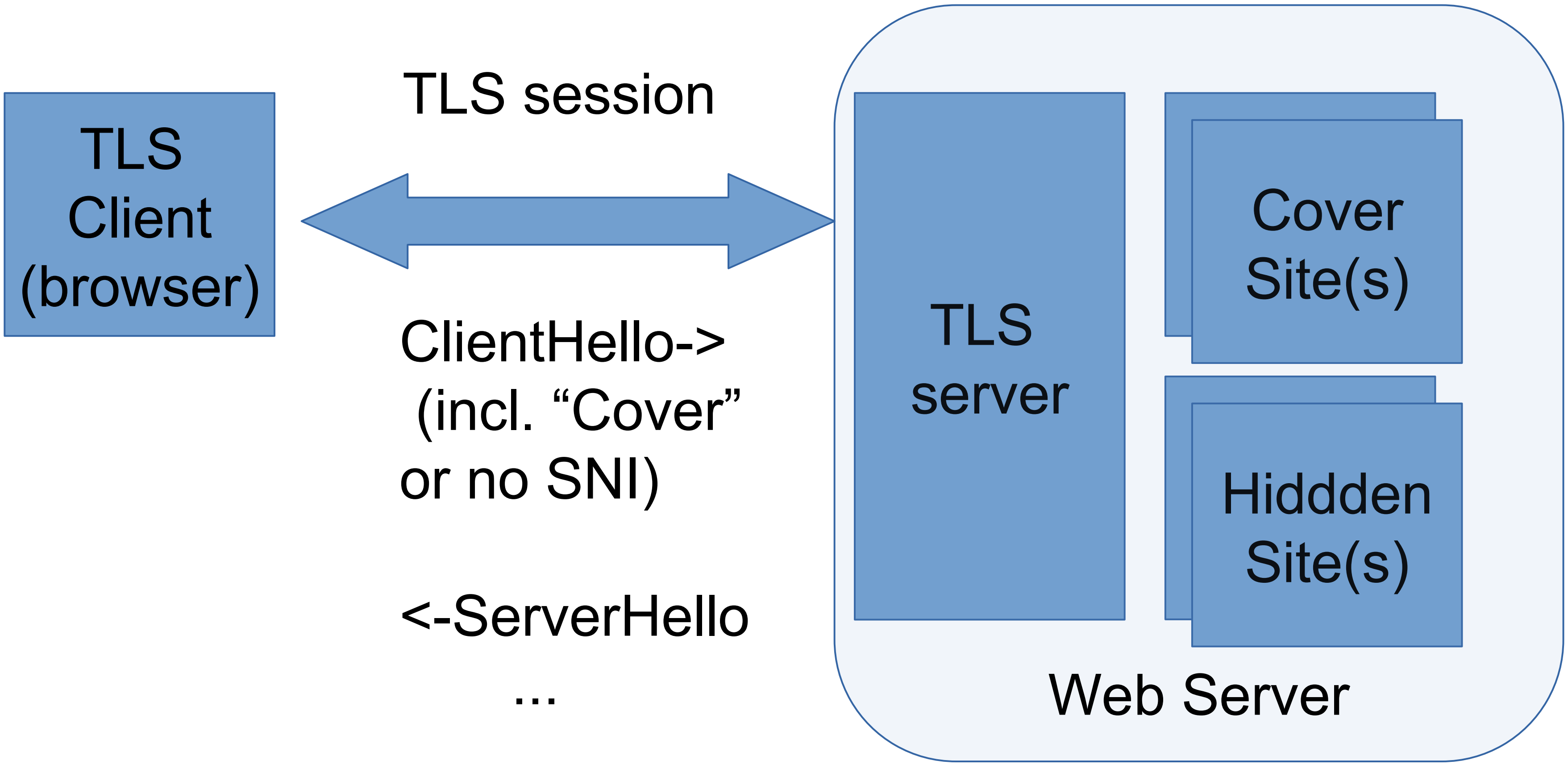
SNI as a leak

- Domain “fronting” (where the SNI has the hoster’s name but the HTTP request has the “real” DNS name) worked, but was brittle and got turned off by service providers
- Encrypting SNI might be attractive to enterprises that like the idea of hiding the fact that someone is accessing payroll.example.com vs. marketing.example.com
- Previously, we weren’t motivated to try address this quite tricky problem because...
  - TLS server certificate is in the clear prior to TLSv1.3
  - DNS name in clear via Do53, but now we have DoT/DoH
  - Reminder: Remember, we **all** love DoH:-)

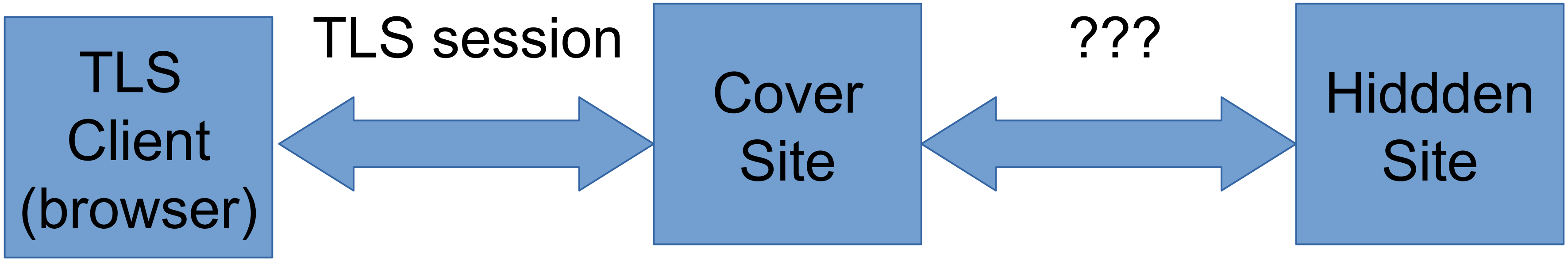
Consensus on the problem

- The IETF’s TLS working group has reached consensus on the problem to be addressed: https://tools.ietf.org/html/draft-ietf-tls-sni-encryption
- Ideal solution likely not achievable, but would:
    - Withstand TLS session message replay to expose SNI
    - Not require a new key distribution infrastructure
    - Make it hard to spoof “cover” server
    - Not create new DoS vectors
    - Not “stick out” (allow hiding in crowds)
    - Provide forward secrecy for encrypted SNI
    - Support co-located or separated “cover” and “hidden” services
    - Work for more than the web (SMTP/TLS etc.)

What we'd like, and can do now ("co-located" variant)



“Split” variant



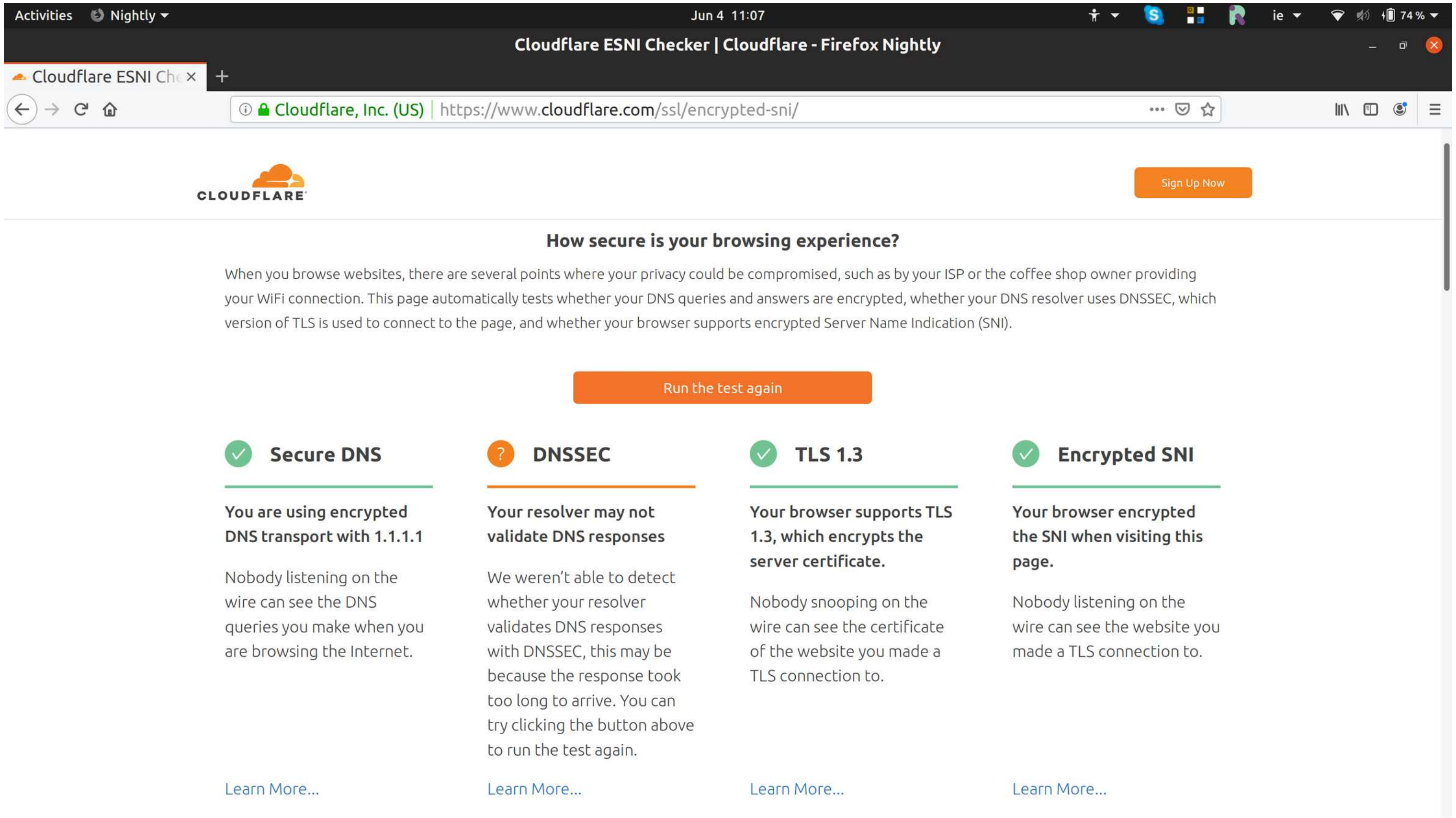
- Details of “cover” to “hidden” protocol are TBD
- Protocol details likely fairly obvious
- Correlation attack likely hard to mitigate
- Terminolgy isn’t final:
  - Cover vs. Front vs. public_name
  - Hidden vs. Private name

  all used variously for the same things

Encrypted SNI (ESNI) or Encrypted ClientHello (ECHO)

- Solution being developed in the IETF TLS WG: https://tools.ietf.org/html/draft-ietf-tls-esni
  - Current draft is version -06; most implementers currently seem to be on draft -02
  - Earlier versions were called ESNI, name changed to ECHO in March 2020 (draft-06)
- Maybe <12 months from being done
- Plenty of interest, so may get fairly wide deployment
- Deployments and Implementations:
  - Cloudflare (https://encryptedsni.com)
  - Firefox nightly
  - Mine:-)
- You can access any Cloudflare-hosted site using ESNI today, e.g. the IETF’s public web site (https://www.ietf.org/)

Encrypted SNI (ESNI)



How secure is your browsing experience?

When you browse websites, there are several points where your privacy could be compromised, such as by your ISP or the coffee shop owner providing your WiFi connection. This page automatically tests whether your DNS queries and answers are encrypted, whether your DNS resolver uses DNSSEC, which version of TLS is used to connect to the page, and whether your browser supports encrypted Server Name Indication (SNI).

✓ **Secure DNS**

You are using encrypted DNS transport with 1.1.1.1

Nobody listening on the wire can see the DNS queries you make when you are browsing the Internet.

? **DNSSEC**

Your resolver may not validate DNS responses

We weren't able to detect whether your resolver validates DNS responses with DNSSEC, this may be because the response took too long to arrive. You can try clicking the button above to run the test again.

✓ **TLS 1.3**

Your browser supports TLS 1.3, which encrypts the server certificate.

Nobody snooping on the wire can see the certificate of the website you made a TLS connection to.

✓ **Encrypted SNI**

Your browser encrypted the SNI when visiting this page.

Nobody listening on the wire can see the website you made a TLS connection to.

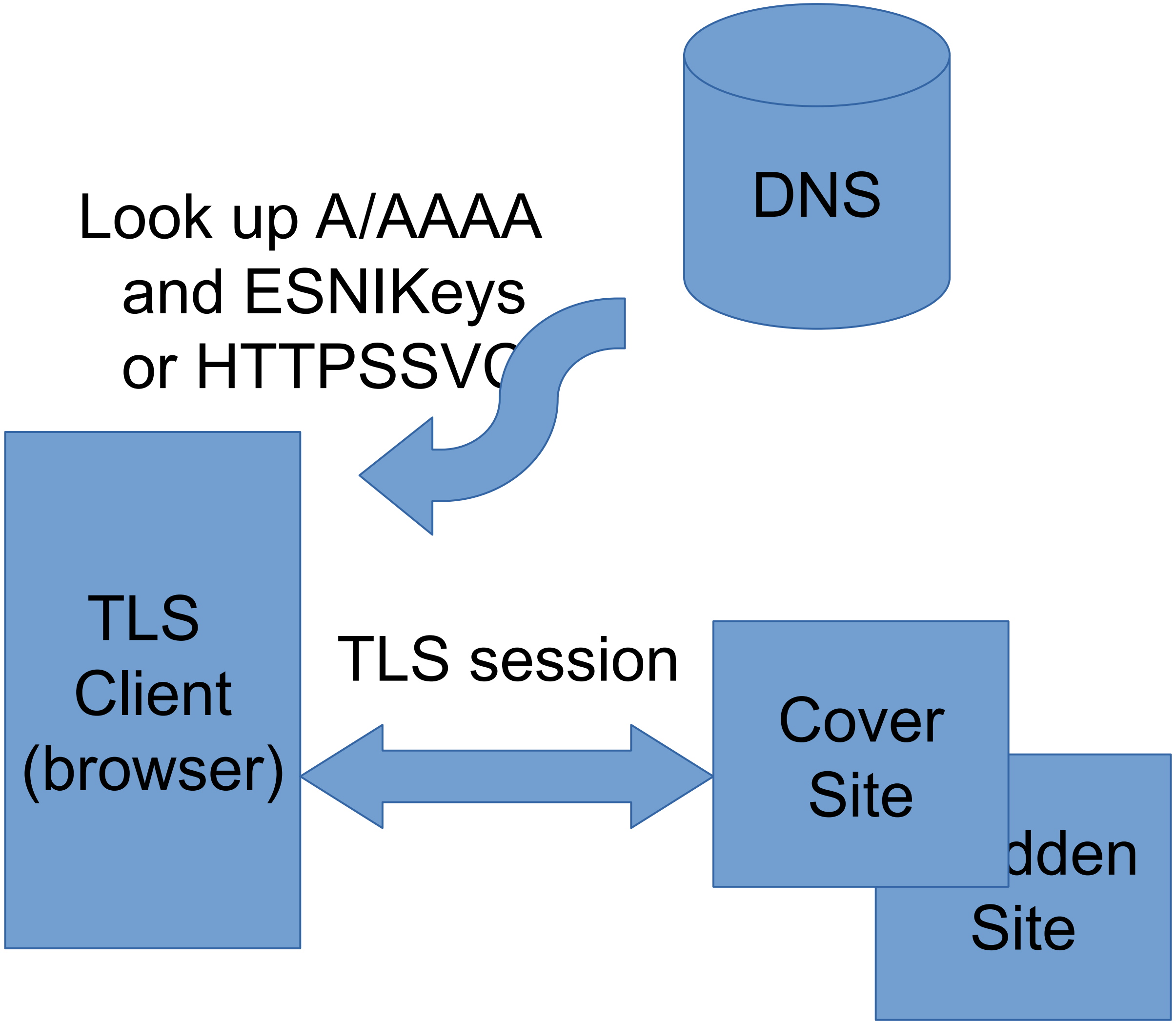
My OpenSSL fork implementation

- I’m working on an implementation of ESNI for the OpenSSL library: https://github.com/sftcd/openssl/tree/master/esnistuff
- Code for versions -02 and -03 and -04 today (March 2020)
- Not tested, no guarantees, early days, etc., etc.
- Test server available at https://defo.ie/
  - Has lighttpd, apache and nginx deployments all supporting ESNI using my OpenSSL fork and v. modest changes to the web server builds
  - Updates ESNI keys hourly, with 3 hour ESNI key lifetime
  - Zone is DNSSEC-signed too btw:-)
  - If you wanna play, let me know, some people have already
- Whether or not any of my code ends up in a standard release of the OpenSSL library is for later and will be determined via the OpenSSL project’s usual mechanisms for handling contributions
- This project is funded by the OpenTech Fund (https://opentech.fund/) and is called “Developing ESNI for OpenSSL” or DEfO and details are also at https://defo.ie

How does ESNI/ECHO work?

- Requires ability to create/consume new DNS resource records
- MUST use TLSv1.3 (earlier versions send server cert in clear so no point)
- DNS privacy (DoT/DoH) not strictly needed but if you don’t, what’s the point?
  - Firefox implementation couples the DoH and ESNI/ECHO
- Web site publishes a new public key/value in the DNS (in “ESNIKeys” or “HTTPSSVC” RR for ECHO)
- TLS client (e.g. browser) checks if DNS record exists
- If it does, all going well, use public value to derive a new shared-secret and send the true server name value in encrypted form in a new TLS ClientHello extension
- The fact that this is being used is visible, but GREASE may help
- TLS client may or may not send an outer/cleartext SNI (“cover” name) as well as the encrypted one
- With ECHO, additional “inner” ClientHello fields can be encrypted, e.g. ALPN

ESNI/ECHO Picture



DNS

Look up A/AAAA
and ESNIKeys
or HTTPSSVC

TLS
Client
(browser)

TLS session

Cover
Site

dden
Site

ESNIKeys (1)

ESNIKeys invented by TLS implementers, so defined using TLS presentation syntax

Draft-02:

```
struct {
        uint16 version;
        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;
```

Draft-03:

```
struct {
        uint16 version;
        uint8 checksum[4];
        opaque public_name<1..2^16-1>;
        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;
```

ESNIKeys (2)

- Draft-02 used TXT RR with bas64 encoding of ESNIKeys
  - published at _esni.hidden.example.com
  - ESNIKeys.version 0xff01
- Draft-03 uses a new (not yet officially assigned) RRTYPE
  - Binary value published at apex of hidden.example.com
  - ESNIKeys.version 0xff02
  - Includes public_name (cover name) **and** address_set
- Model is to frequently re-publish new values in DNS
  - CF currently do it ~hourly, so do I with defo.ie
- Changes to all this are ongoing

Key Derivation (up to draft-04)

- Public key/value from DNS is an (EC)DH public value
- TLS client invents it’s private/public value in same group
- Z is the derived DH secret
- Zx = HKDF-Extract(0, Z)
- ESNIContents couples ESNIKeys, the new client key share and TLS session
- key = HKDF-Expand-Label(Zx, "esni key", Hash(ESNIContents), key_length)
- iv = HKDF-Expand-Label(Zx, "esni iv", Hash(ESNIContents), iv_length)
- ClientESNIInner = nonce + padded version of “hidden” name
- encrypted_sni = AEAD-Encrypt(key, iv, ClientHello.KeyShareClientHello, ClientESNIInner)
- TLS server decrypts then demonstrates knowledge of nonce via an EncryptedExtension

GREASEing

- The fact that ESNI/ECHO is being used is visible
- That may be countered via “greasing” - having browsers that are not using ESNI/ECHO sometimes send a ClientHello that looks like it does use ESNI/ECHO
- Greasing is an anti-ossification TLS implementation trick – clients and servers include garbage values for optional things in order to decrease the probability that middleboxes fixate on currently deployed protocol options – RFC 8701 describes how GREASE works in general for TLS

ESNI and multiple services (1)

- Web sites use multiple CDNs and may switch between those (for price or availability reasons) frequently
- DNS information is cached - all values from the DNS have a time-to-live (TTL) value, which can be small (e.g. 5 seconds) or large (e.g. 1 week)
- When playing the ESNI game, client needs both the IP address of the site (A/AAAA) and the new public key (ESNIKeys)
- Those could get out of whack for various reasons, perhaps especially if a web site is switching between CDNs
- BUT – we don’t know how bad this problem may be – early evidence seems to show it’ll not be a big deal for clients, but could occasionally be for web sites
- If problems were too common, browsers/CDNs might not deploy

ESNI and multiple services (2)

- Two proposals have been made to address this problem:
  - Include IP address information in the ESNIKeys value (in draft-03)
  - Include IP address-range information in the ESNIKeys value
- If draft-03 ends up as the final approach, it'll cause some sysadmins to freak (again:-) as it'd duplicate address information in different places in the DNS, likely breaking tooling and network monitoring
- Address-range proposal seems better to me, but is more complex and maybe has more cases where the A/AAAA and ESNIKeys values might not match - effect could be failed connections or falling back to cleartext SNI when ESNI ought be usable
- Implementers are just starting to play with this so it's not yet set off alarm bells... But it will if it stays as-is;-)

ESNI beomes ECHO

- Problems:
  - “Bespoke” ESNI key derivation was clunky
  - Binding ESNI to outer handshake is a bit tricky (got to consider cut’n’paste attacks of various kinds)
  - There are other things in the ClientHello we might like to hide
  - ESNIKeys DNS RR is a little hard to justify and is very specific to this feature so might be less likely to be deployed
- Solution (plan):
  - Use a more generic/standard key wrapping/derivation that may be useful elsewhere (HPKE)
  - Use an entire “inner” ClientHello as plaintext instead of just SNI, so we can hide more and be more future proof (e.g. hiding ALPN is good too)
  - Work with browser/DNS people on a more generic HTTP alternative services DNS RR construction that also solves other problems (e.g. CNAME at zone apex) to increase liklihood of deployment

HPKE – Hybrid Public Key Encryption

- HPKE aims to be a generic way to “encrypt application data using an AEAD, to a public key”
- Not intended to be an application itself, but rather a tool that protocols and applications can use
- Specification: https://tools.ietf.org/html/draft-irtf-cfrg-hpke
- (My) code: https://github.com/sftcd/happykey
  - OpenSSL based, intended for integration with my ESNI-enabled OpenSSL fork

HTTPSSVC Resource Record

- Aims to enable a web site operator to publish various useful bits of information about that web site in the DNS
- Specification: https://tools.ietf.org/html/draft-ietf-dnsop-svcb-httpssvc
- Problems being addressed:
  - Public key for ECHO
  - What ALPN value(s) to use?
  - Is the origin also at some alternative host:port as well?
  - CNAME at zone apex problem for CDNs

Encrypted inner ClientHello

- Model for ECHO becomes:
  - Create an “inner” CH with the “real” settings, including the SNI that we want
  - Encrypt that “inner” CH (using the public key from HTTPSSVC) and include the ciphertext as an extension in the “outer” CH
  - Make up the “outer” CH to look “innocent” (probably by copying the non-sensitive bits of the “inner” CH to the “outer”)
- Issues that creates:
  - More complexity (sigh)
  - Client has to guess (via trial decryption) if server’s answer relates to inner or outer CH (e.g. if wrong/old ESNI public key used)

ECHO Status

- Latest ESNI draft (-06) sort of specifies ECHO
- More work needed before we can do interop, so likely draft-07 needed soon
  - I’m doing (some of) that work now:-)
- A branch of my OpenSSL fork that had a guess of how that might end up (done before the draft-06 published)
  - https://github.com/sftcd/openssl/blob/encch/esnistuff/echo.md
- Padding discussion provides a good example of the kind of complexities involved
  - https://github.com/tlswg/draft-ietf-tls-esni/pull/209

Conclusion

- Encrypted SNI or ECHO is a fine continuation of our efforts to reduce cleartext meta-data exposing privacy-sensitive information
- Spec definitely a work-in-progress but it also definitely works
- Interest levels high so reasonable to expect stuff will happen in 2020
- Feel free to get in touch if you're interested

  stephen.farrell@cs.tcd.ie

  draft spec: https://tools.ietf.org/html/draft-ietf-tls-esni

  (my:-) code: https://github.com/sftcd/openssl/tree/master/esnistuff

  standalone HPKE code: https://gitbhub.com/sftcd/happykey

  DEfO project site: https://defo.ie/

- Thanks!