

# Encrypted Client Hello (ECH) a TLSv1.3 work-in-progress (April 2023)

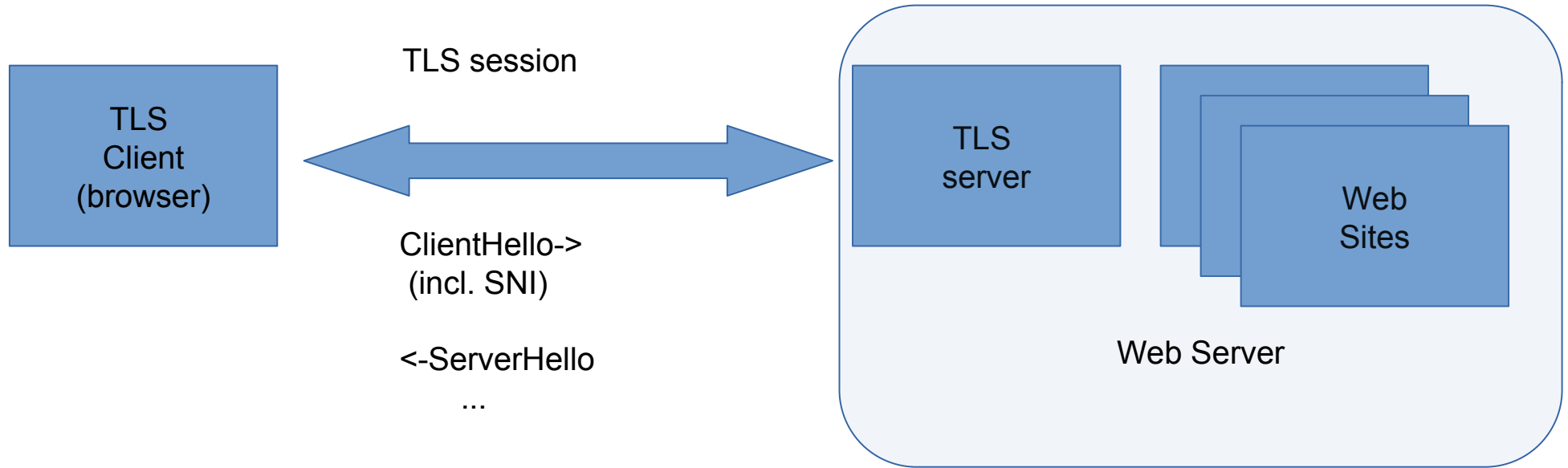
# Overview

- As well as knowing about today's Internet it might be useful to know a bit about what's coming and how some things evolve
- So I'll describe some work I've been helping with for the last couple of years, how that might go and how it might ultimately affect you
- That's a thing called "Encrypted ClientHello" (ECH)

# 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 VirtualHost) can, and very frequently does, serve multiple web sites
- Each of those may (and is likely to) use different TLS server key pairs/certificates, which are the things that allow a TLS client (like your browser) to authenticate the web site, i.e. to know that you're really connecting to someone who (is related to someone who) controls "tcd.ie"

# TLS for multiple web sites

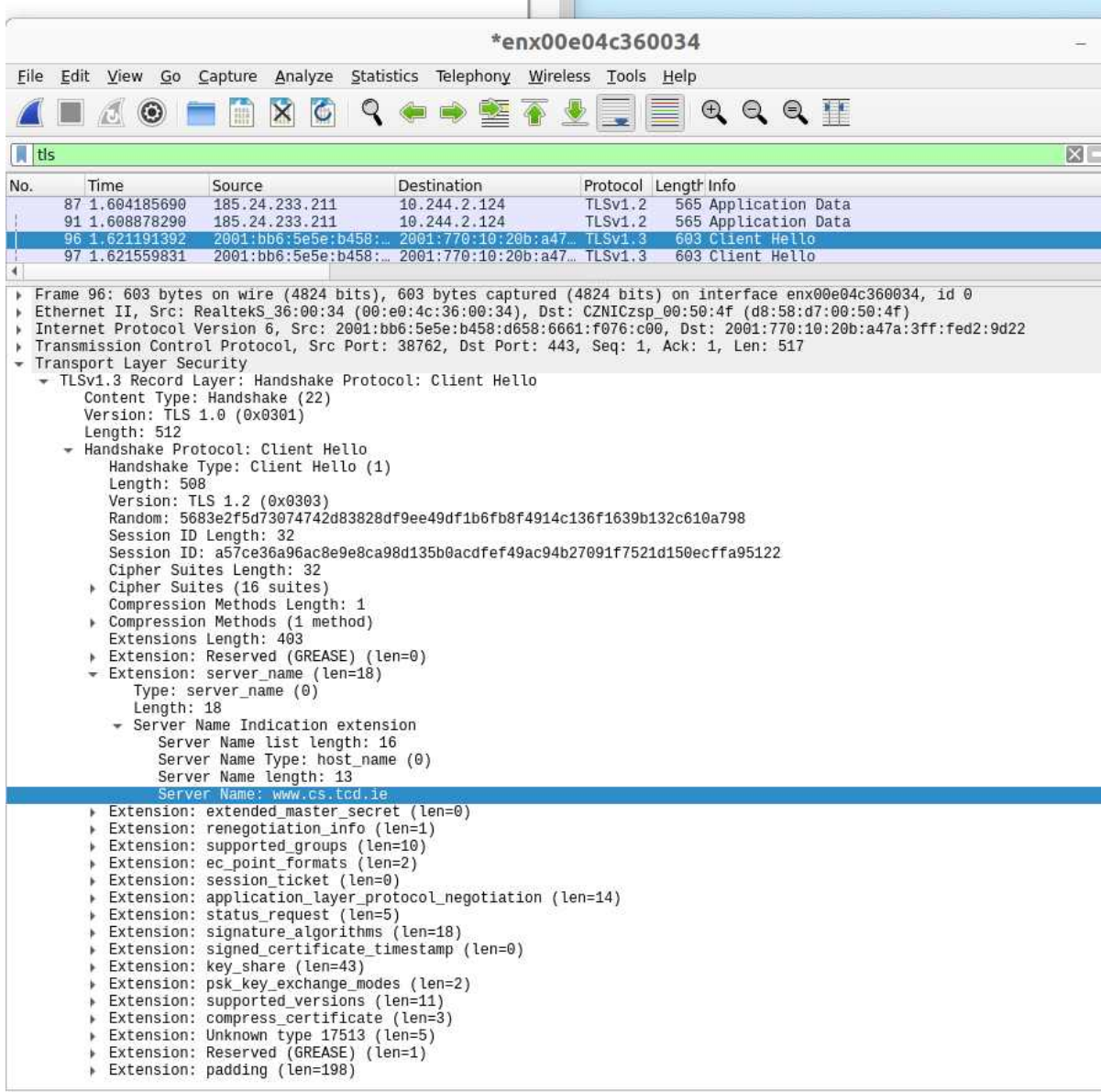


# TLS and SNI

- 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)

# The SNI “Leak”

- Wireshark knows how to decode a ClientHello and we can see the cleartext SNI value on the right
- That ClientHello message was sent from a browser when I accessed <https://www.cs.tcd.ie>



# 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 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

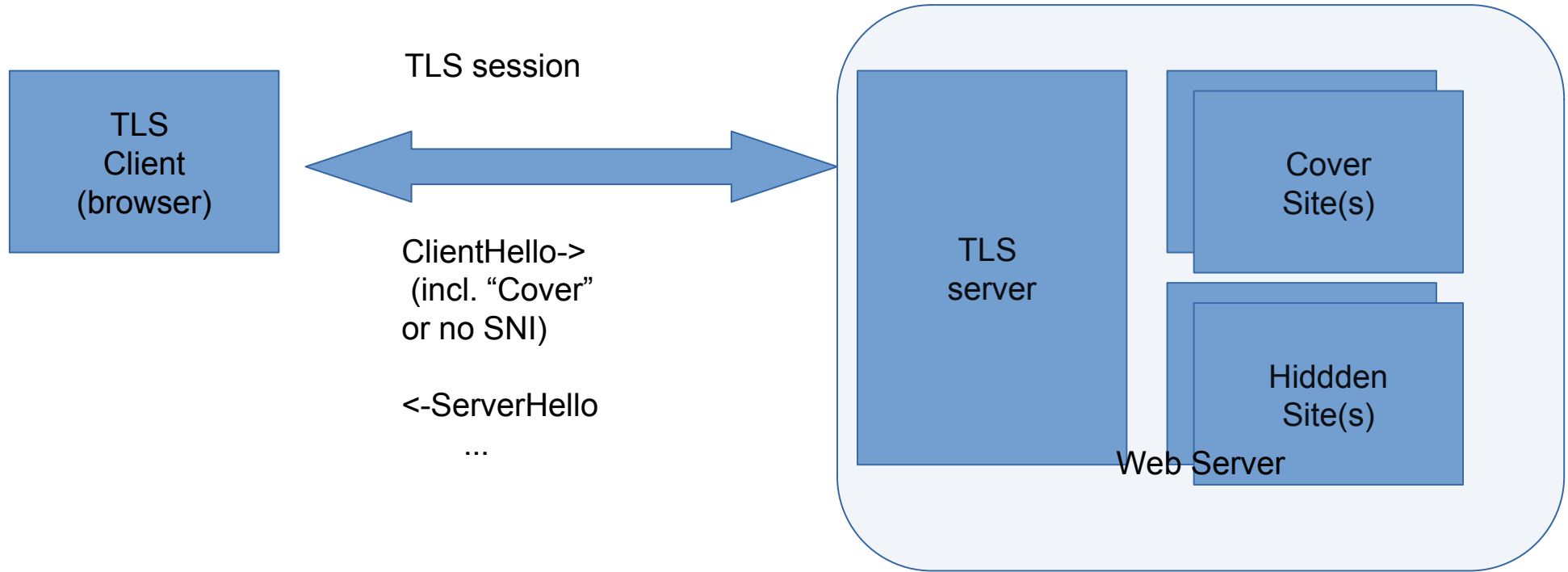
<https://www.bleepingcomputer.com/news/security/south-korea-is-censoring-the-internet-by-snooping-on-sni-traffic/>

# SNI as a leak

- Domain “fronting” (where the SNI has the hoster’s name but the HTTP request has the “real” DNS name) was brittle and got turned off by service providers
- Previously, we weren’t motivated to try address this quite tricky problem because...
  - TLS server certificate was sent in the clear prior to TLS1.3
  - DNS name sent in clear using DNS protocol, but now we do have DNS privacy mechanisms (DoT/DoH)



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



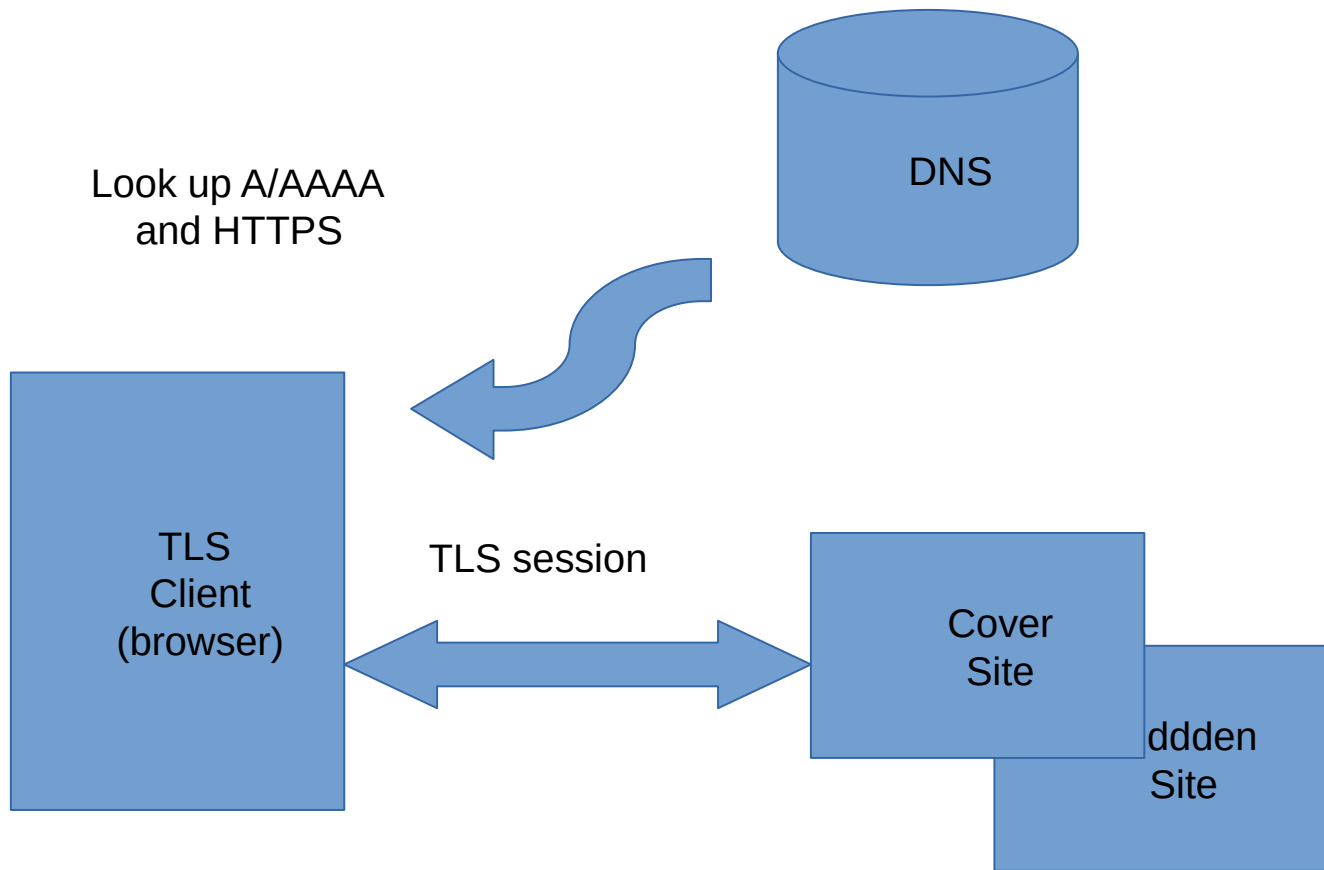
# Encrypted ClientHello (ECH)

- Solution being developed in the IETF TLS WG:  
<https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni>
- Current draft is version -16, in development since mid-2018, with the spec “stable” since Aug 2021
- Latest version seems complete, now at the stage of testing
- Multiple implementations exist, including mine
  - <https://github.com/sftcd/openssl/> is my “fork” of OpenSSL - a very widely used TLS/encryption library that can be used with apache, nginx, lighttpd (web server implementations) and haproxy versions I’ve modified to support ECH
  - List: <https://github.com/tlswg/draft-ietf-tls-esni/wiki/Implementations>

# How does ECH work?

- Needs ability to create/consume new DNS resource records
- Needs TLS1.3 (earlier versions send server cert in clear)
- DNS privacy (DoT/DoH) not strictly needed but if you don't use that maybe there's less point in using ECH (Browsers will likely couple the two)
- Web site publishes a new public key/value in the DNS ("SVCB" or "HTTPS") with some additional keys (HPKE public values) for ECH
- ECH-aware client (e.g. browser) can check if DNS record exists and has ECH keys
- All going well, use those ECH keys with HPKE (RFC 9180) to derive a new shared-secret and send the "real" ClientHello message encrypted inside an "outer" ClientHello message
- The fact that ECH is being used is still visible

# ECH Picture



# GREASEing ECH

- The fact that ECH is being used is still visible
- That may be countered via “GREASEing” - having browsers that are not using ECH sometimes send a ClientHello that looks like it does use ECH
- GREASEing 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 - RFC8701

# The details....

- Hybrid Public Key Encryption (HPKE – RFC9180)
- HTTPS or SVCB RR in DNS
- Inner/Outer ClientHellos
- HelloRetryRequest gank
- Code

# HPKE – RFC 9180

- A way to do ephemeral-static ECDH for a payload, i.e., to agree a symmetric key and use that to AEAD encrypt a payload that's usually a symmetric key (used to encrypt the inner ClientHello)
- Handles the various algorithm identifiers (KEM, KDF, AEAD), mixing the additional data as required and how to do multiple encryption operations (and decryptions too:-)
- My implementation was merged to OpenSSL in Nov 2022
- Used for ECH, but also other new protocols: MLS, OHAI, an extension to QUIC, ...
  - <https://datatracker.ietf.org/doc/rfc9180/referencedby/>
- A bit tedious to code, but not too hard and good test vectors exist and it seems to provide a useful abstraction
- The public key values that enable ECH are the static HPKE values that are intended to be published in the DNS... as an `ECHConflgList`
  - The relevant TLS server has to have the associated static private values

# ECHConfigList

```
opaque HpkePublicKey<1..2^16-1>;
uint16 HpkeKemId; // Defined in RFC 9180
uint16 HpkeKdfId; // Defined in RFC 9180
uint16 HpkeAeadId; // Defined in RFC 9180
struct {
    HpkeKdfId kdf_id;
    HpkeAeadId aead_id;
} HpkeSymmetricCipherSuite;
struct {
    uint8 config_id;
    HpkeKemId kem_id;
    HpkePublicKey public_key;
    HpkeSymmetricCipherSuite cipher_suites<4..2^16-4>;
} HpkeKeyConfig;
```

```
struct {
    HpkeKeyConfig key_config;
    uint8 maximum_name_length;
    opaque public_name<1..255>;
    Extension extensions<0..2^16-1>;
} ECHConfigContents;
struct {
    uint16 version;
    uint16 length;
    select (ECHConfig.version) {
        case 0xfe0d: ECHConfigContents contents;
    }
} ECHConfig;
ECHConfig ECHConfigList<1..2^16-1>;
```



# A simple ECHConfigList

Base64 encoded:

```
AD7+DQA65wAgACA8wVN2BtscOl3vQheUzHeIkVmKIiydUhDCliA4iyQRCwAEAAEAAQALZXhxbXBsZS5jb20AA  
A==
```

Output of `base64 -d | hd:`

```
00000000  00 3e fe 0d 00 3a e7 00  20 00 20 3c c1 53 76 06  |.>...:... .<.Sv.|
00000010  db 1c 3a 5d ef 42 17 94  cc 77 88 91 59 8a 22 2c  |...:].B...w..Y.",|
00000020  9d 52 10 c2 96 20 38 8b  24 11 0b 00 04 00 01 00  |.R... 8.$.....|
00000030  01 00 0b 65 78 61 6d 70  6c 65 2e 63 6f 6d 00 00  |...example.com..|
```

# HTTPS and SVCB RRs

- Publishing ECHConfigList values in the DNS is fine but not enough to motivate deployment, we needed to try solve some other problems (for CDNs) to get there
- In particular multiple-CDNs and “CNAME at apex” problems, but also other web issues (e.g. port! =443, HSTS, alt-svc, ALPN)
- The result is new SVCB and HTTPS RR types
- HTTPS is a simplified version of SVCB
  - <https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-svcb-https>
- In addition to ECHConfigList value, an SVCB can also contain “hints” about IPv4 or IPv6 addresses, “alias” and ALPN information – the idea is to (ideally) get everything a browser might need in as few DNS queries/answers as possible
  - The goal is to avoid additional round-trips – browsers care most about speed!
- Situations requiring chasing SCVB RRs are envisaged (sort of like CNAME chasing)

# A simple HTTPS RR value

- QNAME is `foo.example.com`
- Zone file fragment is

```
foo.example.com. 10 TYPE65 \# 71  
( 000100000050040003efe0d003ae7002000203cc153760  
6db1c3a5def421794cc778891598a222c9d5210c2962038  
8b24110b000400010001000b6578616d706c652e636f6d0  
000 )
```

# Rotating ECH Keys

- While HPKE is ephemeral-static, servers may (and do) rotate the values
  - Hourly for my servers and cloudflare's
- Servers need to keep “old” private values and be able to decrypt using those, as well as the freshest ones currently visible in DNS
  - In my case, I keep the 3 most recent, and for twice the TTL
- TLS servers (e.g. web server) therefore need to regularly reload keys and DNS infrastructure needs to be setup to handle regular re-publication of HTTPS/SVCB values
  - Working on <https://datatracker.ietf.org/doc/draft-ietf-tls-wkech/> to cater for different administrators for web servers and DNS

# Browser Actions

- A DoH- and ECH-enabled browser will likely send DNS queries for A, AAAA and HTTPS RRs (and maybe SVCB too) before initiating contact
  - It may then chase SVCB RRs as indicated in the relevant values
  - Likely that HTTPS/SVCB will only be used if browser knows queries sent via DoH
- If there is an ECHConfigList in a relevant RR then the browser will supply that to some TLS client API, before connection is attempted
  - nss for FF, boringssl for chrome etc.
- When the TLS client library initiates the TLS handshake and has a (valid-looking) ECHConfigList then the library will trigger use of ECH

# TLS Client Library Actions for ECH

- Prepare the “inner” ClientHello (CH) as usual expressing the “real” intent
- “Compress” the inner CH where the same values will also be present in the outer CH
- Lots of complexity there for any value fed into the key derivation now we have to prepare for two outcomes (the client doesn’t know ECH will work out as planned)
- More complexity due to e.g. “custom” CH extension handlers
- Use an HPKE API to prepare and encrypt the inner CH, producing an ECH extension value to include in the outer CH
- Prepare the outer CH so that it would also work (if ECH fails at the server for whatever reason)
- Ready the state machine so that expected responses to either inner or outer CH can be handled
- Send the CH (including an ECH extension) to the server...

# Web Server Actions

- Provide configuration settings for loading (and re-loading) ECH keys and ECHConfigList values
  - Might be tied to specific VirtualHosts or similar
- Don't peek into the SNI in the ClientHello as it might be an outer CH when the real value to be used is in the inner CH
  - Can require (fairly simple) server code changes

# TLS Server Library Actions

- If configured for ECH, check very early if there is an ECH extension in CH
- If so try decrypt (using HPKE) and if successful replace state with content of inner CH
- Handle error cases by falling back to processing outer CH
- If ECH processing succeeded then signal that to client in the ServerHello random value using a “magic” value



# Complications...

- Handle GREASE so things don't fail
- HelloRetryRequest (HRR) where initial CH (inner or outer) has parameters (in particular chosen curves) incompatible with server, but that may be recoverable
- Early-data used in 0RTT – will be encrypted with inner CH keys so can't be handled if ECH decryption fails (and needs a GREASE'd outer psk extension)
- Split-mode ECH where different servers handle front-end and back-end
  - Note: split-mode + HRR can require major s/w architecture changes in front-end (at least for haproxy)
- Clients like curl or wget will require significant re-engineering to handle the complexity of HTTPS/SVCB RRs and related caching – without ECH they can just use the OS's DNS or a simple cache of A/AAAA values

# Code

- My OpenSSL fork with ECH is at <https://github.com/sftcd/openssl> current dev branch is “ECH-draft13c”
- API design doc: <https://github.com/sftcd/openssl/blob/ECH-draft-13c/doc/designs/ech-api.md>
- Last time I looked there were about 8k new lines of code but affecting lots of different parts of the TLS state machine
- TLSv1.3 already involved a fairly major re-factor of the OpenSSL code
  - The developers sensibly re-used lots of minor internal APIs
  - Changes that affected those were the trickiest to figure out and I may have messed up badly, it’s quite hard to tell for sure
- That’s likely a major work-item (and risk) for the upstream OpenSSL maintainers to incorporate
  - And they’ll for sure disagree with some internal API detailed design changes I’ve made along the way
  - My changes also had to deal with the evolution of the upstream code over the couple of years involved, which weren’t insignificant
- The changes for web servers (apache, nginx, lighttpd) were very easy
- Haproxy is a bit harder (esp for split-mode which isn’t finished)
- Getting curl to work wasn’t hard but doing it right (with full SCVB/HTTPS support) would be a lot more work
- Pointers to all that code at <https://defo.ie>

# Testing...

- Basic interop tests
  - On localhost, with other client/server implementations - needs lots of logging and time
  - Lots of scripting, some non-trivial
- Test harnesses
  - OpenSSL `make test` target is complex
  - Have added ECH API tests to that, (lots) more TBD
- Fuzzing is still TBD
- Internet-accessible servers (e.g. <https://defo.ie/>) with key rotation needed  
DNS capability

# Personnel

- It'd likely have been much harder to do any of this without contacts with other relevant developers
  - It's important to try to work well with others
- Some contact points for ECH for other code-bases changed over time and different folks' timelines and priorities also change over time
  - Don't assume your priority or timeline is everyone's
- Dealing with funding is a thing too, but it's good to have the discipline of a contract for work like this that extends over years
  - Thanks to <https://www.opentech.fund/>

# Interesting thing #1

- People who dislike DNS privacy, probably dislike ECH even more; for them, this is browsers and major web sites taking away information from which they've benefited
  - They used to monitor DNS queries but now DoT/DoH make that much harder
  - Switched to monitoring SNI from TLS ClientHello messages, and now ECH may kill that off
- They may have been using that information for what you consider good or bad (they considered it “good”):
  - Censorship, net-nanny, corporate policy enforcement, whitelisting TLS sessions to not MITM
- That DNS or SNI information could also be used for profiling or sold to advertisers, but I'm not aware of reliable information that that has happened

# Interesting thing #2

- The people proposing/backing ECH are (I believe) doing so to try improve privacy (e.g. me, mozilla, aclu)
- ECH may however further increase centralisation since hiding in larger crowds is more effective (Cloudflare reportedly front for about 7.5M web sites!)
- It could be that the likes of Cloudflare (test server deployed), apple (implemented client and server earlier) and Google (implemented in boringssl library and chromium) are the main beneficiaries of ECH – how should we consider that result?

# Interesting thing #3

- ECH may make some kinds of Internet measurement harder, maybe a lot harder
- How do we try get accountability for software and systems without leaving open gaping holes to be exploited by attackers or nation-states?
  - Answer: we don't know (or at least I don't)

# Interesting thing #4

- My guess is that whether or not, and how well, Google chrome implement ECH GREASEing will be the key determinant of whether or not ECH gets long term deployment
- As ECH sticks out, censors like the GFW can just block it
- That's easy if it's not widely used and would kill deployment
- If almost all ClientHello messages appear to contain ECH then such blocking gets harder
  - Maybe not for the GFW but perhaps for most other censors



# Last interesting thing

- The processes for defining and implementing ECH are open to anyone with the skills and interest to get involved
  - Consumes a lot of time, but getting the skills required isn't really hard, all of you are certainly smart enough
- So you could (if you choose) end up involved in key decisions about how the Internet affects us all that can reverberate up to fairly highly political levels
  - (Don't get a big head though:-)
  - That's part of the “permissionless innovation” I mentioned before
- It's also possible to be as involved with fewer technical skills, e.g. in civil society organisations or campaigning groups

# Want to try it out?

- As of 2023/03...
- For a recent FF
- Enable DoH (settings/General/Network-settings and scroll down to “Enable DNS over HTTPS”) with your chosen provider
- GOTO about:config
  - Set network.dns.echconfig.enabled to True
- Try it with some test sites:
  - <https://defo.ie/ech-check.php> (mine)
  - <https://crypto.cloudflare.com/cdn-cgi/trace> (cloudflare’s – not as nice IMO:-)
- There are similar instructions for chromium-based browsers
- I hope, in future, it’ll be a standard thing you’ll never notice

# Conclusion

- ECH is an attempt to close off a remaining privacy leak in TLS
- ECH is not that simple, but that's because it's a hard problem (to encrypt before 1<sup>st</sup> contact, at scale)
- We're now at a point where larger scale test deployments are possible
- It'll be interesting to see if this gets widely deployed or not in the next few years
  - I hope it does!