A well-known URL for publishing ECHConfigLists

https://datatracker.ietf.org/doc/draft-ietf-tls-wkech/

Stephen Farrell, Rich Salz, Ben Schwartz

IETF 115, London, November 2022

28/10/22 16:00:19

Changes since IETF 114

- draft-00 described an actual running prototype operated by Stephen Farrell on https://defo.ie for its colocated domains.
- draft-01 has been redesigned to serve a broader range of ECH deployment architectures.
  - Still only contains the information necessary for ECH, but extensible to other SVCB parameters.
  - Not actually implemented at all.
  - **Seeking WG input on this new design**

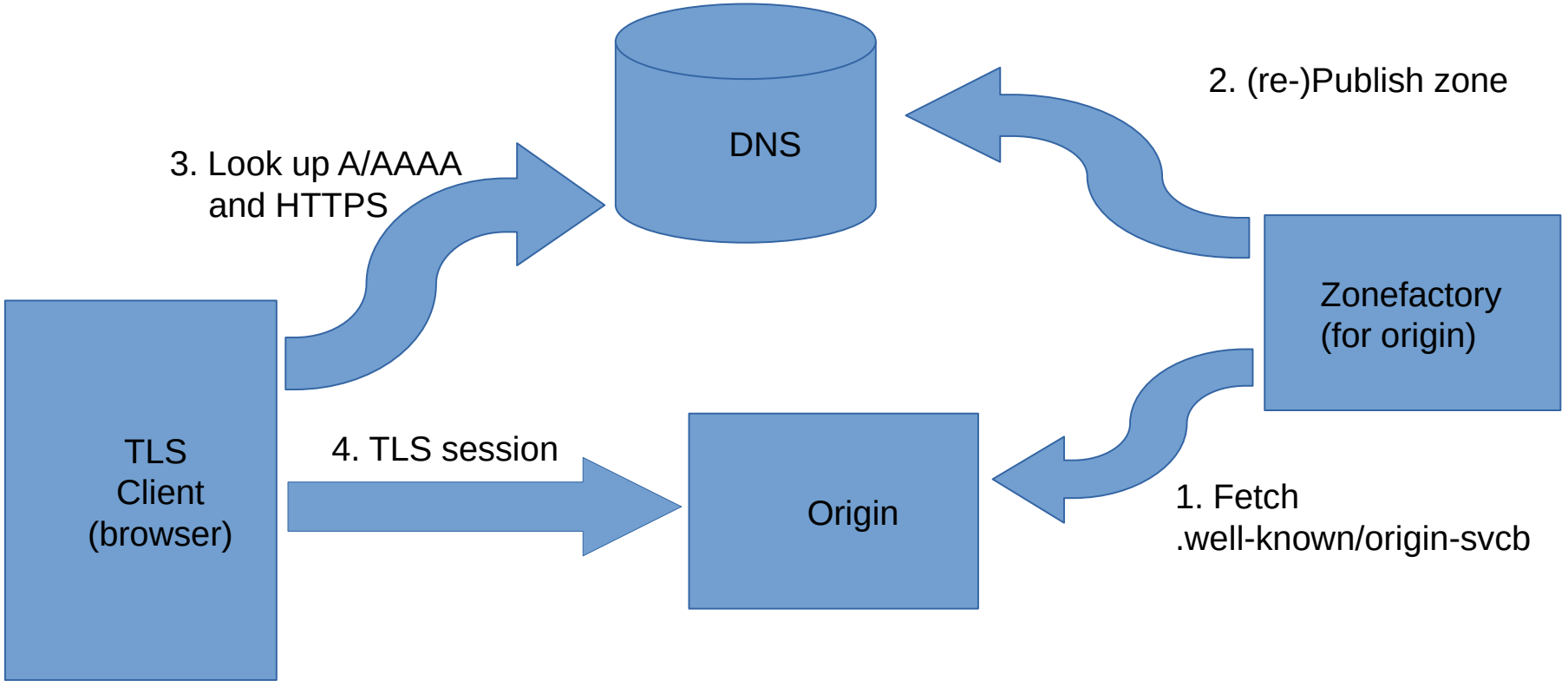
Background

- Encrypted ClientHello (ECH) normally relies on HTTPS records to publish the public key (ECHConfigList) in the DNS.
- It’s easy (and tempting!) to paste the public key into your zonefile and declare success.
- This breaks key rotation, especially if your customers are pasting your key into zones you don’t even control.
- **Goal**: Make proper dynamic zone generation **easier than doing it the wrong way**.

Technical Summary

- Specifies a protocol between an **origin** and a **zone factory** to keep the DNS zone up to date whenever a server rotates its ECH keys.
- The zone factory already knows the origin’s name, IP addresses, and any SVCB parameters other than the ECH keys.
- The origin hosts a JSON blob containing either:
  - a list of “endpoints” (1:1 with ServiceMode HTTPS records).
  - an instruction to alias this origin to some other origin.
- Hosted at /.well-known/**origin-svcb** by default.
  - This proves that the contents are authoritative for this origin.

Picture


DNS
2. (re-)Publish zone
3. Look up A/AAAA
and HTTPS
Zonefactory
(for origin)
TLS
Client
(browser)
4. TLS session
Origin
1. Fetch
.well-known/origin-svcb

Comparison: -00 vs. -01

version -00

```
[{
  "desired-ttl": 1800,
  "ports": [443, 8443],
  "echconfiglist": "ABC..."
}]
```

version -01

```
{
  "endpoints": [{
    "port": 443,
    "ech": "ABC..."
  }, {
    "port": 8443,
    "ech": "XYZ..."
  }]
}
```

Resulting HTTPS RRs

There are two ways to use this JSON:

1. If the zone factory is generating RRs from scratch:

```
origin.example. IN 1800 HTTPS 1 . ech=ABC... port=443
                         HTTPS 2 . ech=XYZ... port=8443
```

2. If the zone factory is starting with template RRs:

```
                         HTTPS 1 . alpn=h2,h3 port=443
                         HTTPS 2 . alpn=h2 port=8443
```

then it adds each ECHConfigList to the template with a matching port number:

```
origin.example. IN 1800 HTTPS 1 . alpn=h2,h3 port=443 ech=ABC...
                         HTTPS 2 . alpn=h2 port=8443 ech=XYZ...
```

Aliasing example

Origin JSON

```
{

  "alias": "cdn.example"

}
```

The "alias" and "endpoints" options are independent of ECH "shared mode" and "split mode". "alias" is recommended if the origin has the same SvcParams as the public name.

Zone Factory Output

Could be any of:

- `HTTPS 0 cdn.example.`
- `CNAME cdn.example.`
- `HTTPS 1 cdn.example. [parameters copied securely by DNSSEC from cdn.example.]`
- `...`

  (Templates are not supported for aliasing)

Multi-CDN Example

CDN JSON:

```
{
  "endpoints": [{
    "ech": "BBB..."
  }]
}
```

Origin JSON:

```
{
  "endpoints": [{
    "priority": 1,
    "ech": "AAA..."
  }, {
    "priority": 1,
    "target": "cdn.example.",
    "ech": "BBB..."
  }]
}
```

Zone Factory Output

```
HTTPS 1 . alpn=h2,h3 ech=AAA..

HTTPS 1 cdn.example. alpn=h2 ech=BBB...
```

- The origin dynamically incorporates https://cdn.example/.w-k/origin-svcb into its own JSON output, so the ECH keys stay up to date.
- The zone factory is configured statically with templates containing the other parameters.
- The “priority” and “target” identify which ECHConfigList goes with which template.

Other notable details

- DNS TTL is chosen by the zone factory, but MUST be less than the HTTP freshness lifetime.
  - Hard to figure out using simple HTTP client APIs...
- Ordinary web clients “SHOULD NOT” try to use this in lieu of real HTTPS records.
  - Not very effective, plus it creates a supercookie.

To Be Determined

- Exact template matching rules
  - or should we just stuff the whole HTTPS record into JSON?
- Static bootstrap IP requirement for zone factories
  - Otherwise the zone factory could lose access to the origin permanently due to a bad config push.
- Support for non-HTTPS protocols
  - Seems straightforward but harder to set up.

What do you think?