DNS Privacy for Clients
Requirements, Challenges, and Solutions

DNS Client to Resolver Privacy via Encryption

The original specifications for the Domain Name System (DNS) specified unencrypted transport protocols, specifically UDP and TCP.
As a result of the disclosure of pervasive monitoring, the IETF has sought to address privacy in updates to standards including DNS.
While there are now two competing privacy standards for DNS, DNS over TLS (DoT) and DNS over HTTPS (DoH), both of those standards are underpinned by the same encryption standard, Transport Layer Security (TLS).

We will look at DoT, as it is literally DNS over TLS, the bare minimum of technology to support privacy for DNS communications from a client to a DNS resolver.

Objective: Upgrade Existing Transport

The primary conceptual goal is to preserve any and all existing policies, topologies, relationships, and configurations, and upgrade the transportation mechanism to achieve privacy via encryption.

DNS Resolver: No Name Required (by Standards)

For several reasons, DNS resolvers have never actually needed to have a DNS name. The primary reason is that there would be a “catch-22”: A client cannot learn the IP address for a DNS resolver from its name, without first using a DNS resolver at a known IP address. Even if there were a name known, the DNS protocol only uses IP addresses for processing recursive requests.

TLS: Server Name is Mandatory

In contrast, the TLS protocol starts with a well-defined “handshake”, where clients and servers exchange information, including the TLS certificate(s) of the server. The client must validate the certificate, and one mandatory attribute for a certificate is the value of “server_name”. It must be a syntactically valid domain name (even if it does not resolve in DNS), and must not be a literal IP address (IPv4 or IPv6).

The CAs Problem

TLS as a protocol, specifies the format for certificates, known as X.509, along with particular extensions to that protocol. The most common use of TLS is the World Wide Web (WWW), where TLS certificates are issued by a number of Certification Authorities (CAs). Each CA has its own “Root CA” certificate is used to sign certificates that the CA issues. Any certificate can be used to sign other certificates, and thus certificates can form a “tree” where the signatures included in a certificate, can be validated against the public key of the parent (signing) certificate. The public keys of the set of Root CAs that are generally trusted, are published, and included in WWW web browsers (e.g. Safari, Chrome, Firefox.)

This introduces several problems. First is the set of CAs required for validating a certificate. The second is that such WebPKI CAs are only allowed to sign certificates with valid DNS names (real names under real TLDs). Third is that the process of signing a certificate is somewhat complex and requires Internet connectivity. Fourth is that (at least until recently) certificates have to be purchased. Fifth is the limited duration and expiration for certificate CA signatures.

DoT: No Standard for Name Discovery

The last element causing problems for the “upgrade” objective, is that there is no standard defined for mapping an IP address to a resolver’s DNS name. This is a dual problem, since resolvers are not required to even *have* a DNS name.

DANE TLSA for Certificates

DNSSEC Authentication of Named Entities (DANE) is an IETF standard for securely publishing information in DNS. The specific DNS record types include TLSA, a TLS certificate “fingerprint”, which is signed via DNSSEC. This provides ways of validating TLS certificates by verifying their fingerprints via DNSSEC signatures.

TLSA records provide a method of validating either the server certificate itself, or any certificate above the actual server certificate in the chain of signatures present in the certificate “bundle”. TLSA has two subsets of validation methods: one is stand-alone, where validating the certificate fingerprint (or the fingerprint of another certificate above it in the chain); the other is a dual-validation method, which also requires that the regular WebPKI validation must succeed.

The use of TLSA record types that do not require CA validation, enables the use of non-public-CA issued certificates, i.e. self-signed certificates or private-CA signed certificates. This relaxes the naming requirement of using a legitimate top-level domain (TLD). This also eliminates the need for public CA transactions, and allows for certificate re-issuance ad infinitum. In the case of private CA usage, it prevents the risk of the same certificate being issued by multiple CAs.

Reserved Name Space

The DNS name space is anchored at the top with aptly named Top Level Domains (TLDs) including country-code TLDs (CCTLDs). The CCTLDs are reserved for issuance by the ISO (international standards organization). The ISO administers all of the 2-letter names according to their naming scheme for countries, ISO 3166-1. That scheme happens to also reserve a number of 2-letter codes for “local use”, which is guaranteed not to be used for a country code, and thus will never be a real TLD. Such a namespace is “zz”, such as “example.zz”.

“Hello, My Name is _____”

A resolver’s clients, by their very nature, have delegated DNS resolution to the resolver, which does so on behalf of its clients. This means that any private-use name that the resolver gives to itself, would not exist *except* internally to the resolver (and for its clients in the context of their relationship with the resolver.)

If the resolver was contacted over an unencrypted connection, and the client asked for information within such a local namespace, the resolver would be able to provide the necessary information to the client. The necessary information about the resolver is:

- The IP address(es) of the resolver
- The TLSA for the resolver’s TLS certificate
- The necessary DNSSEC keys (trust anchors) for the zones involved
- Resolver type (resolver or forwarder)

It would then be possible for the client to establish a TLS connection to the resolver which would pass the TLS certificate validation process. (DNSSEC is needed to ensure data integrity.)

The only remaining problem is discovery of the name that the resolver uses.

Clients and Forwarders and Resolvers, Oh My!

One of the hidden challenges is that the name server used by a client (whether statically configured or learned via some mechanism like DHCP), may not actually be doing resolution, but may only be forwarding the DNS queries to another server.

This mode of operations is known as “forwarding”, and the server doing that is commonly referred to as a “forwarder” as opposed to a “resolver”.

It is possible for forwarders to be chained, with a client talking to a forwarder, which forwards to another forwarder, etc., until the last forwarder forwards to a resolver. Each chain will always terminate at a full resolver (or fail to resolve queries).

NB:

- A client cannot tell directly that it is talking to a forwarder.
- A server (forwarder or resolver) is unable to distinguish a normal client from a forwarder.

One new DNS mechanism, EDNS (extended DNS), allows for new metadata to be exchanged from a client to a server. However, EDNS is explicitly only supported across one such DNS “hop”, from a client to a server. There is no ability for a client to have any EDNS metadata passed from a forwarder to a resolver. EDNS metadata will only ever return information reported by the first server, i.e. the first forwarder in a chain if forwarders are used.

There is one thing that will always be passed by a forwarder through a chain of forwarders to a resolver: an in-band DNS query (for any name, class, type 3-tuple).
If the query is successful, the answer will be returned to the client.

This is the basis for the proposed discovery mechanism: asking the resolver to identify itself via in-band DNS queries.

Islands of Security: DNSSEC Trust Anchors

DNSSEC supports the notion of an “island of security”. This is a DNS “zone” which does not have a secure delegation from above, from a DNSSEC signed zone. The DNSSEC validation of an island of security, requires that the validator have installed a corresponding DNSSEC Trust Anchor (TA). A TA consists of a DNS name (the name of the zone) and the DNSSEC parameters including the public key, the key’s algorithm and hash algorithm.

The main problem with Trust Anchors is having a method of publishing them, which is itself secure. Any secure method works, including putting them on a well-known HTTPS web page, or publishing them in another DNSSEC signed zone as a KEY record.

Discovery: Security from Nothing

Without using some *existing* trust anchor (DNSSEC or CA set), it is literally impossible to discover anything that will be secure, and thus trustworthy. Is that a fatal flaw, or a temporal (time-based) issue?

The main issue has to do with whatever is standardized on the discovery side, and what capabilities are reasonable to require for a client. Given that this is entirely new, there is no specific requirement to support anything existing. The measure of “what is reasonable” is thus open to exploration.

Context: “You Are Here”

There are two primary environments where a resolver is required for use by a client: wired networks; and wireless. By their nature, wired networks tend to have fixed infrastructure components that do not change frequently, even if the set of clients may come and go. Wireless networks themselves may be mostly fixed, but are difficult to tell apart, particularly when private IP addresses (RFC 1918) are used, and thus the network numbers are not unique. It is often the case that a mobile client will connect to multiple wireless networks in its lifetime, that have the same IP address range. (It is actually highly unlikely that any mobile client will not encounter such duplication ever.) WiFi SSIDs may be useful for disambiguation, but are not a secure method of determining identity (i.e. they can be spoofed trivially.)

Excluding DNS Resolver discovery itself, it would be reasonable to expect that the client software would be able to track some other attributes of the network infrastructure to which it connects, and to associate discovered resolvers with the corresponding infrastructures, even when those are not guaranteed to be unique.

If the resolver names *themselves* are generated in a way that guarantees their global uniqueness, it then becomes feasible to attempt to determine whether a network connection *should* have a particular resolver on it, and thus avoid re-discovery, and/or to reject resolvers attempting to fool the client into using them.

On the wired network with mostly unchanging infrastructure, things such as MAC addresses, VLAN IDs, IP addresses, for routers, resolvers, and other devices, may facilitate identification of previously known networks, and thus independent recall of the resolver name the client knows.

This reduces the insecure element of discovery to a very small window when the client first performs whatever discovery process is needed, after which the client may be able to use (and re-use) the trust anchor(s) needed to secure the TLS connection to the resolver.

A Well-Known Name for Publishing

Since there is no standard mechanism for discovery of resolver names (or even of establishing resolver names), there is a need to establish one. The choice of name is itself arbitrary, other than selecting an appropriate DNS location for it. The “arpa” TLD is appropriately controlled and there is precedent for placing names like this under arpa; thus **resolver-name.arpa** is proposed.

Requirements for Upgrade to DoT, Summarized

Since the goal is to upgrade a client from regular DNS to DNS over TLS (DoT), the starting point is the IP address of the existing server (forwarder or resolver).

The client and resolver can both be presumed to have some new components (e.g. a software package, or scripts to configure unmodified software that might need to be installed).

All of this follows from the underlying needs of TLS, DNSSEC, scalability, and no presumed coordination between clients and resolver(s). An ideal solution would enable resolvers to upgrade themselves in a manner that is as secure as possible, requiring little to no technical expertise.

Additionally, managed upgrades to resolvers can benefit from a one-and-done client configuration, where a set of resolvers could share some security property, that would enable clients to move between networks which are served by those resolvers, without having to redo the untrusted element of the upgrade.

The requirements are:

- A randomly generated resolver name (in the reserved 2-letter TLD “zz”)
- A self-signed (or private CA signed) certificate with this resolver’s name (for TLS)
- A matching DNSSEC signed zone with this name
- A TLSA record with the fingerprint of the certificate public key in this zone
- **resolver-name.arpa** as a DNSSEC signed zone, with NS name of the resolver’s name
- A KEY record *in* **resolver-name.arpa**, with content identical to the KSK of the zone
- The DNSKEY record (KSK) of **resolver-name.arpa** is the overall trust anchor
- Both zones are served authoritatively, on the resolver (or via equivalent means)

TOFU-Secured Upgrade, Unless Configured Separately

The proposed method (trust on first use, aka TOFU) is only necessary if no other mechanism is available to securely establish a TLS connection from the client to the resolver. If another method is available, it should be used. This will typically be some managed configuration coordinated between resolver and clients, or a securely published means of disseminating configuration to end-users (such as QR codes).

The “first use” refers to obtaining and remembering the entry point, which is the DNSKEY record for the **resolver-name.arpa** zone on the resolver. This becomes the trust anchor, and managing it is an important part of the security posture of the client. This includes managing key updates that might occur (using RFC 5011 methods), and if necessary, comparing the first-learned dates of trust anchors.

After this, queries to the resolver should be attempted using the DoT configuration exclusively. *Reconnection* to an apparently-known network should initially be attempted via DoT on the learned unique name. Learning new resolvers’ names may be performed if the client determines this is a different network. Implementations might choose to do this only with user approval.

Simple Example: Upgrade Resolver to offer DoT

Generate unique name: **abcd1234efgh.zz**

Generate self-signed TLS certificate for **abcd1234efgh.zz**

Calculate fingerprint for TLS certificate for **abcd1234efgh.zz**

Generate DNSKEYs for **abcd1234efgh.zz**

Set A for **abcd1234efgh.zz**

Set AAAA for **abcd1234efgh.zz**

Set TXT for **server-type.abcd1234efgh.zz** to “resolver”

Set TLSA for **_853._tcp.abcd1234efgh.zz** to certificate fingerprint
(Underscore values means “DNS over TLS”)

Sign zone **abcd1234efgh.zz**

Generate DNSKEYs for **resolver-name.arpa**

Set NS of **resolver-name.arpa** to **abcd1234efgh.zz**

Set KEY for **abcd1234efgh.zz.resolver-name.arpa** to KSK of abcd….zz

Sign zone **resolver-name.arpa**

Configure resolver to listen on DNS over TLS (port 853 TCP) using certificate

Configure resolver to serve zones **resolver-name.arpa** and **abcd1234efgh.zz**

READY TO UPGRADE CLIENTS TO DoT

Simple Example: Upgrade Client-Resolver to DoT

Client:	Resolver:
get DNSKEY for **resolver-name.arpa**	
	(Return DNSKEY)
Save new DNSKEY, use from here forward	
get NS for **resolver-name.arpa**	
	(Return NS == unique name) NS **abcd1234efgh.zz**
Validate NS using DNSKEY get KEY for **abcd1234efgh.zz.resolver-name.arpa**	
	(Return KEY)
Validate KEY using DNSKEY	
get DNSKEY for **abcd1234efgh.zz**	
	(Return DNSKEY)
Validate new DNSKEY using KEY	
Save new DNSKEY, use from here forward, validate all returned data	
get A for **abcd1234efgh.zz**	
	(Return A if exists)
get AAAA for **abcd1234efgh.zz**	
	(Return AAAA if exists)
get TXT for **server-type.abcd1234efgh.zz**	
	(Return “resolver”)
get TLSA for **_853._tcp.abcd1234efgh.zz** (Underscore values means “DNS over TLS”)	
	(Return TLS certificate fingerprint)
Connect over TLS. Validate TLS certificate with TLSA.	

DNS over TLS Upgrade Complete (PRIVATE)

Variation 1: Private CA

The primary reason that Certification Authorities exist is to sign certificates for third parties. The validity of a certificate which has been signed by a CA root certificate, relies on a well-known trusted public key for the CA's root certificate. The operation of the actual server using its certificate, requires that the server present its certificate, prove it has possession of the private key matching the public key of the certificate (by signing some client-supplied data using the private key), and that the certificate's signature can be validated by the public key of the CA certificate.

NB: In the case of many public CAs being trusted, any of those CAs can sign the certificate, meaning that more than one certificate, each signed by a different CA, might allow for a substitute certificate to be possible. In the case of only a single private CA being trusted, this possibility is foreclosed, and the CA signature is actually much more secure as a result.

The use case for a Private CA would be the management of multiple individual networks within some larger entity, such as in the "franchise" model of operations, but with the tight operation of the DNS and certificate elements involved. Each franchisee would issue their own certificate requests and store the certificates and private keys, but would rely on their parent franchise operator to run the CA and the primary DNS servers. The parent would need to know the names and IP addresses of the various servers to populate the DNS zones. The franchise CA would sign the certificates, allowing the use of TLSA records which have the fingerprint of the CA (signing) certificate, rather than the server certificates themselves. The franchisee resolver would transfer the signed DNS zones from the franchise.

This simplifies the zone contents, and facilitates the use of managed DNS zones. The resolver no longer is required to operate the DNSSEC components, reducing the technical requirements on the resolver and on the operation thereof. Certificate revocation is handled via DNS (TLSA record removal), and no DNSSEC keys leave the franchise.

In addition, the DNSSEC trust anchor for all franchisees can be the same, which is the key-signing key (KSK) for the **resolver-name.arpa** zone instances. Each network location will have its own unique instance (version) of *resolver-name.arpa*, with its respective unique name of the resolver, certificate, KEY record, and IP address(es). The zone-signing key (ZSK) for each instance can be unique, but having the shared KSK means only one trust anchor is required for clients, even if they connect to networks at different franchisees.

The client gets the name from the resolver (under resolver-name.arpa), validates it via the trust anchor and DNSKEY records for resolver-name.arpa, then gets the KEY record, the records in the uniquely named zone, including the TLSA record for the resolver itself. The TLSA record has the CA certificate's fingerprint, so the certificate validation is done indirectly using both the CA certificate and resolver certificate in the certificate "bundle". Then, the client connects over TLS, and private DNS queries can be done. All of this is validated using either TLS or DNSSEC public keys, anchored and the trusted record for the franchise. This trust anchor is learned when the client connects to the first franchisee location, but is then trusted and known, and used, whenever visiting any subsequent new franchisee location.

Since franchisees do not possess any of the DNSSEC private keys or the CA private key, they cannot operate the resolver without the support and permission of the franchise.

Variation 2: No CA, Managed Trust Anchor

The strength and scalability of DANE is more apparent when no CA is used, but where a managed trust anchor is used to delegate control and authority for a set of resolvers.

The use case for this would be the trusted federated management of individual networks within some larger entity, such as in the "branch office" model of operations, with the independent operation of the DNS and certificate elements involved. Each office would issue their own self-signed certificates and store the certificates and private keys, and would also manage their DNS zones. The head office would managed the primary trust anchor, the key-signing key (KSK) for **resolver-name.arpa** only, and use that for signing each office's zone-signing key (ZSK).

In this model, the need for coordination and communications between the branch offices and the head office is minimized, while the technical capabilities of each branch office is more significant. The branches are able to operate independently outside of the ZSK signing, which can be done ahead of time in bulk to facilitate a greater degree of autonomy for the branch offices.

Most other aspects of DNS in this are similar to the "private CA" model. The DNSSEC trust anchor for all branches is the same. Each network location will have its own unique instance (version) of *resolver-name.arpa*, with its respective unique name of the resolver, certificate, KEY record, and IP address(es). The zone-signing key (ZSK) for each instance will be unique, but having the shared KSK means only one trust anchor is required for clients, even if they connect to networks at different branch locations. The zone instances are signed by the ZSKs, and only the DNSKEY sets (ZSKs and KSKs combined) are signed by the KSK (thus why it is called the "key-signing key"). The DNS zones are operated by the branch, and served from the resolver itself (or from a site-specific authority server).

The client gets the name of the resolver from the resolver (under resolver-name.arpa), validates it via the trust anchor and DNSKEY records for resolver-name.arpa, then gets the KEY record, the records in the uniquely named zone, including the TLSA record for the resolver itself. The TLSA record has the *resolver name's* fingerprint, so the certificate validation is done *directly* using only the resolver certificate. Then, the client connects over TLS, and private DNS queries can be done. All of this is validated using either TLS or DNSSEC public keys, and the trusted single KSK record (the trust anchor) for the organization. This trust anchor is learned when the client connects to the first location, but is then trusted and known, and used, whenever visiting any subsequent new location of the organization.

This does require more trust in the branch offices, since the time frame for revoking the ZSK is dependent on the KSK signatures, which can be significant. The difference between this and the "private CA" mode has no impact on the client's trust, validation, or performance, and the client would not generally be aware unless examining the actual certificate structure of the resolver's certificate.

Variation 3: Private CA Hierarchy

This is basically a variant of the “Private CA”, to be used if the scale of a single CA is exceeded, or for some other related reason (number of keys, etc.)

It involves creating some number of intermediate CA certificates, for signing the server certs, and signing those intermediate certificates with the private CA. It’s basically an extra level of indirection, or possibly more levels. The root CA is still the certificate whose fingerprint is used in the TLSA records, and the certificate bundles would require the root CA, the intermediate signing certs, and the resolver’s own cert.

The use case for this would be the loose management of individual networks within some larger entity, such as in the “franchise” model of operations, but with the tight operation of the DNS and certificate elements involved. Each franchisee would issue their own certificate requests and store the certificates and private keys, but would rely on their parent franchise operator to run the CA and the primary DNS servers. The franchise CA would sign the certificates, allowing the use of TLSA records which have the fingerprint of the CA (signing) certificate, rather than the server certificates themselves. The resolver would transfer the signed DNS zones from the franchise.

This simplifies the zone contents, and facilitates the use of managed DNS zones. The resolver no longer is required to operate the DNSSEC components, reducing the technical requirements on the resolver and on the operation thereof. Certificate revocation is handled via DNS (TLSA record removal), and no DNSSEC keys leave the franchise.

In addition, the DNSSEC trust anchor for all franchisees can be the same, which is the key-signing key (KSK) for the **resolver-name.arpa** zone instances. Each network location will have its own unique instance (version) of *resolver-name.arpa*, with its respective unique name of the resolver, certificate, KEY record, and IP address(es). The zone-signing key (ZSK) for each instance can be unique, but having the shared KSK means only one trust anchor is required for clients, even if they connect to networks at different franchisees.

The client gets the name from the resolver (under resolver-name.arpa), validates it via the trust anchor and DNSKEY records for resolver-name.arpa, then gets the KEY record, the records in the uniquely named zone, including the TLSA record for the resolver itself. The TLSA record has the CA certificate’s fingerprint, so the certificate validation is done indirectly using the CA certificate, all intermediate certificates, and resolver certificate in the certificate “bundle”. Then, the client connects over TLS, and private DNS queries can be done. All of this is validated using either TLS or DNSSEC public keys, anchored and the trusted record for the franchise. This trust anchor is learned when the client connects to the first franchisee location, but is then trusted and known, and used, whenever visiting any subsequent new franchisee location.

Since franchisees do not possess any of the DNSSEC private keys or any of the CA hierarchy private keys, they cannot operate the resolver without the support and permission of the franchise.

What Not To Do

Do not re-use the resolver name, or DNSSEC keys, or certificate private keys.

Resolver Name

Do not re-use the resolver name.

Even if the parameters for the resolver name are all the same, this is a bad idea, precisely because the only parameter that should be re-used is the IP address. The certificate keys and DNSSEC keys should be unique per resolver, so that the risk of compromise is minimized, and the impact of compromised keys is also minimized.

Furthermore, if a compromise is discovered, the number of resolvers which need to have their keys replaced is “one”, rather than “a bunch” or “all of them”. A compromise of a key has the effect of compromising the key for every duplicated resolver instance, and they would all need to be replaced.

DNSSEC Keys

Do not re-use the DNSSEC keys. Anyone obtaining the private keys, would be able to forge the DNSSEC signatures for any zone which shares the keys. This forgery would not be detectable, since detection requires observation. Nothing short of querying every resolver on the planet is guaranteed to succeed, and then only if and when the forgery is active.

Certificate Keys

Do not re-use certificate private keys. Anyone obtaining the private keys on any instance would be able to forge every certificate that used those keys. If an attacker can forge a certificate, or rather serve a valid certificate on a different server, the only additional requirement would be for spoofing DNS or poisoning a DNS cache. DNSSEC only provides protection to validating resolvers, non-validating resolvers can still be poisoned, allowing the attacker to appear to be the valid server.

Why Would You, Anyways?

There is no benefit to re-using anything, and lots of benefit to having unique names and keys for every resolver, with the possible exceptions described previously (Private CA, or Common KSK). Having the ability to uniquely identify the resolver instances is extremely helpful from an operational and support perspective, and for limiting the impact of any potential security issues.

Forwarders With **resolver-name.arpa**

Do not deploy this discovery mechanism on forwarders.
Or, in the alternative, use the methods below so that clients are able to fully discover the upstream resolver(s), and determine whether this forwarder uses DoT.

Forwarders Are Not Resolvers

The discovery mechanism is intended for the discovery of resolvers, so that DoT can be used for a client to communicate with the resolver privately. A forwarder using **resolver-name.arpa** prevents discovery of its upstream resolver on that name, blinding the client.

If a forwarder were to employ this **resolver-name.arpa** zone technique, the resolver would necessarily be in one of two states: forwarding over unencrypted connections to a resolver (or another forwarder); or forwarding over an encrypted connection (DoT or DoH) to a resolver (or another forwarder). However, without these extra components/techniques, any client of the forwarder would not be able to know which was the case, and would not be able to even determine whether it was talking to a forwarder or a resolver.

Forwarder Upstream Discovery and Publication

The publication mechanism for resolvers, would also be used for forwarders, to advertise their names and other information. However, in the case of forwarders, it is also necessary to advise clients that the server is in fact a forwarder, and the identities and connectivity available to the upstream servers. The discovery mechanism used by a client, can also be used by a forwarder to attempt to upgrade the connection(s) it has to its upstream(s). The exact same process would be used, and depending on the upstream server's status, it would either be possible to upgrade to DoT (or DoH), or it would not.

This is doubly important if any of the upstream servers are themselves forwarders, for all of the reasons listed at the top of this section: the upstream forwarder might not have an encrypted upstream of its own. This forwarder would need to discover the actual status, and include that in the published information about itself.

Upstream Information

The publication by a forwarder needs to be augmented with the list of upstream servers' names, and the relevant details for those each of those servers:

- Trust Anchor (DNSSEC KEY)
- IP Addresses
- Server Type (forwarder or resolver)
- Transport details (e.g. DoT or not)
- TLSA record

The information is published within the zone of the unique name of the forwarder, as child records. The list of upstream servers uses a well-known child name "upstreams". This is a list of PTR records, one per upstream, with the unique name of that upstream.

The child records with the KEY and address records, would have names underneath this forwarder that match their respective names, so if the first upstream had a name of "upstream1.zz", and the forwarder was named "myname.zz", the name for the records would be "upstream1.zz.myname.zz". These records would allow the client to contact "upstream1.zz" directly.

References and Notes

Transport Layer Security (TLS) Synopsis (RFC 5246)

Client determines IP address of Server
Client establishes TCP connection to Server
Client sends a “hello” message including the requested Server_Name (RFC 6066)
Server sends a Certificate matching the Server_Name
Client validates the Server Certificate

DNS over TLS (RFC 7858) Challenges Detailed in DoT Profiles (RFC 8310)

TLS **prohibits** use of IP address literals as Host Name in a Certificate
DoT Strict Security profile satisfied by pre-configured Server Name + IP address
No standard method for learning Server Name from IP address
Server Name to IP requires DNS, only meets Strict Security if signed by DNSSEC
Server Name must be pre-configured regardless
Server Name must resolve via DNS
Certificate validation requires a CA-issued cert plus WebPKI (PKIX),
or DANE TLSA in a securely delegated zone

DNS over TLS Identity/Discovery Problems

Server_Name must be a “real” DNS name (managed and delegated)
DANE requires the Root Trust Anchor (TA), or a secure method of distributing a TA
Opportunistic methods can be downgraded or compromised
Any *pre-configured* solution requires coordination, managed devices, or secure pub.
Largest proportion of resolvers are forwarders, or unmanaged, or both
Clients (esp. software/API) largely cannot be upgraded (easily, quickly, or even ever)

DNS over TLS Identity Pub & Discovery Solutions

Solution Basis: Trust on First Use (TOFU) method
Generate globally unique names and corresponding certs (pub)
Remember learned TA contents, including date/time (discovery)
Get resolver name, TLSA, etc., tie to resolver name as primary key
Track network details and match names+TA(s) to network(s)
Prefer known over unknown, oldest over newer
Run a local (loopback) forwarder from UDP to DoT after discovery/upgrade complete