anoBytes | TECHNOLOGY SOLUTIONS

Set up encrypted, validated DNS on Linux with Quad9

Point systemd-resolved at Quad9 over DNS-over-TLS with strict DNSSEC validation, so your lookups are unreadable on the wire and forged records are rejected. Covers Fedora 44, RHEL 9 and 10, Ubuntu 26.04 LTS, and CachyOS.

QUAD9 · DNS-OVER-TLS · DNSSEC · SYSTEMD-RESOLVED · HOW-TO GUIDE

3 components to combine

DNS-over-TLS, DNSSEC validation, and Quad9's malicious-domain blocking

4 distributions covered

Fedora 44 · RHEL 9 and 10 · Ubuntu 26.04 · CachyOS

A Linux system sends its DNS lookups in plaintext over port 53 unless you change something. Anyone who can watch that traffic (your ISP, the operator of the coffee-shop Wi-Fi, or someone sitting between you and the resolver) can see every domain you ask for, and anyone in a position to modify it can change the answers that come back. This guide points `systemd-resolved` at Quad9 over DNS-over-TLS (DoT: DNS carried inside an encrypted TLS connection) with strict DNSSEC validation, so lookups are unreadable in transit and forged records are rejected. The procedure is the same on all four distributions once `systemd-resolved` is the active resolver; the package names, the desktop network settings, and one RHEL enablement step differ.

Read this first if you run RHEL

SUPPORT STATUS DIFFERS FROM THE OTHER THREE

On Fedora, Ubuntu, and CachyOS, `systemd-resolved` is the default resolver. On RHEL it is neither the default nor installed: NetworkManager writes `/etc/resolv.conf` itself, and `systemd-resolved` ships as a separate package. Red Hat's release notes describe it as an unsupported Technology Preview on RHEL 8, RHEL 9, and RHEL 10.0. The 10.1 and 10.2 notes carry no entry for it either way, so treat the Technology Preview status as current until you have checked otherwise: read the notes for your own minor version, and on a machine under a production support contract, confirm with Red Hat before enabling it. The setup here works on both major versions, and Quad9 validates DNSSEC upstream whatever the host does. Step 1 has the extra work RHEL needs.

PARTS Three pieces that solve different problems

These technologies are complementary, and turning one on does not give you the others.

TECHNOLOGY	WHAT IT PROTECTS	WHAT IT LEAVES ALONE
DNS-over-TLS (DoT)	Confidentiality and integrity in transit. The lookup rides inside a TLS connection on TCP 853, so the network path can neither read nor change it.	Whether the answer is genuine. A resolver that lies, or that has been misconfigured, is still believed.
DNSSEC	Authenticity and integrity of the data itself. Signatures let a validating resolver spot a forged record and fail the lookup rather than return it.	Confidentiality. A DNSSEC-validated lookup sent over port 53 is still readable by everyone on the path.
Quad9 (the resolver)	A recursive resolver, run by a Swiss non-profit foundation, that blocks domains known to be malicious and sends no EDNS Client Subnet (ECS) data, so authoritative servers never learn your rough location.	Your queries, as far as Quad9 itself is concerned. It has to resolve them, so it sees them. See the Limits section.

NIST draws the same division in SP 800-81r3: DNSSEC "does not protect the confidentiality of DNS query/response data", while "DoH, DoT, and DoQ only protect the query/response transaction and do not provide integrity protection and source authentication for DNS data in responses." Used together, DoT keeps the lookup private while it crosses the network, DNSSEC proves the record came from the zone that owns it, and Quad9 refuses to resolve domains on its threat feeds.

FLOW Where the plaintext stops

Plaintext DNS still exists on this machine, and it stays on loopback (`127.0.0.53`), where applications reach the stub listener. Everything that leaves the machine is DNS-over-TLS to Quad9 on 853.

MAP Distribution cheat-sheet

Once `systemd-resolved` is running, the procedure is identical everywhere. These are the only differences:

DISTRIBUTION	RESOLVED BY DEFAULT?	TOOLS FOR STEPS 6 AND 7	DESKTOP GUI (STEP 4C)
Fedora 44	Yes	`sudo dnf install -y bind-utils nmap-ncat tcpdump`	KDE or GNOME
RHEL 9 / 10	No. Install and enable it in Step 1.	`sudo dnf install -y bind-utils nmap-ncat tcpdump`	Usually headless, so use Option A
Ubuntu 26.04	Yes	`sudo apt install -y bind9-dnsutils netcat-openbsd tcpdump`	GNOME
CachyOS	Yes, since September 2024	`sudo pacman -S --needed bind openbsd-netcat tcpdump`	KDE

`dig` comes from `bind-utils`, `bind9-dnsutils`, or `bind`; `nc` from `nmap-ncat`, `netcat-openbsd`, or `openbsd-netcat`. The `resolvectl` checks need no extra packages. The systemd version varies across the set (RHEL 10.2 ships 257, Fedora 44 and Ubuntu 26.04 ship 259, Arch-based systems are on 261 as this is written), so the exact layout of the Step 5 status output differs between them while the values that matter stay the same.

PREP Prerequisites

- One of the distributions above, with administrative access (`sudo`).
- NetworkManager managing the connection. It is the default on desktop Fedora, Ubuntu, and CachyOS, and on RHEL, and Step 4 assumes it.
- Basic comfort editing a configuration file in a terminal.
- A way back in that does not depend on DNS if this is a remote machine. Strict DNS can break name resolution while you are tuning it, so know the server's IP address before you start.

CachyOS: turn off the Welcome app's DoH or DoQ before you start

TWO RESOLVERS WILL COMPETE FOR THE SAME JOB

CachyOS-Welcome has a DNS page of its own. Its DNS-over-HTTPS (DoH: the same idea as DoT, carried over HTTPS) and DNS-over-QUIC (DoQ: carried over QUIC) options install `blocky`, a separate resolver that listens on `127.0.0.1` and takes the query before `systemd-resolved` sees it. Run one of them. If you enabled either option, turn it off first, or both will try to answer and the checks in Steps 5 to 7 will disagree with each other. That page's DNS-over-TLS option behaves differently: it writes the addresses and the certificate hostname onto the NetworkManager connection. That is Step 4 Option B done from a GUI.

01 Confirm (or, on RHEL, enable) systemd-resolved

Fedora, Ubuntu, and CachyOS

`systemd-resolved` is already the resolver here. Confirm it, and confirm that `/etc/resolv.conf` points at its stub:

```
resolvectl status
ls -l /etc/resolv.conf
```

`resolv.conf mode: stub` in the first command's output means `systemd-resolved` is handling DNS. The `Current DNS Server` field shows the upstream resolver it is using (your router now, `9.9.9.9` once you finish); it does not show the local `127.0.0.53` stub listener. The second command should print a symlink such as `/etc/resolv.conf ->`

`../run/systemd/resolve/stub-resolv.conf`; an absolute target, `/run/systemd/resolve/stub-resolv.conf`, is the same thing. Restore it if it is missing:

```
sudo ln -sf /run/systemd/resolve/stub-resolv.conf /etc/resolv.conf
```

On CachyOS, if `systemctl is-active systemd-resolved` prints `inactive` (installs made before September 2024), enable it and restart NetworkManager: `sudo systemctl enable --now systemd-resolved` then `sudo systemctl restart NetworkManager`.

RHEL 9 and 10

Here `systemd-resolved` ships as its own package and is not installed or enabled. Note the support caveat in the box near the top before you proceed. Install it, tell NetworkManager to hand DNS to it, point `/etc/resolv.conf` at the stub, and start it:

```
sudo dnf install -y systemd-resolved
sudo tee /etc/NetworkManager/conf.d/10-resolved.conf >/dev/null <<'EOF'
[main]
dns=systemd-resolved
EOF
sudo ln -sf /run/systemd/resolve/stub-resolv.conf /etc/resolv.conf
sudo systemctl enable --now systemd-resolved
sudo systemctl restart NetworkManager
```

Check the result with `resolvectl status`; it should now report `resolv.conf mode: stub`. Then continue as for the other distributions.

02 Record the current state before you change anything

The settings in Step 3 fail closed, so a typo will stop name resolution. Save the working state first, so you have something to compare against and can confirm the rollback later:

```
resolvectl status > ~/dns-before.txt
```

Step 3 adds a separate drop-in file and leaves the shipped configuration untouched. That is what makes the rollback a single delete. Note your router's DNS address from that output as well: if you need to get back online in a hurry, that is the address you will want.

03 Point systemd-resolved at Quad9 over DNS-over-TLS

Put the settings in a drop-in file. This works the same way on all four distributions:

```
sudo mkdir -p /etc/systemd/resolved.conf.d
sudo tee /etc/systemd/resolved.conf.d/90-quad9-dot.conf >/dev/null <<'EOF'
[Resolve]
DNS=
DNS=9.9.9.9#dns.quad9.net 149.112.112.112#dns.quad9.net 2620:fe::fe#dns.quad9.net 2620:fe::9#dns.quad9.net
FallbackDNS=
DNSOverTLS=yes
DNSSEC=yes
EOF
```

- **Why a drop-in, and why the bare DNS= line comes first**

  This avoids two traps. The shipped configuration is not in the same place on every distribution: Fedora and RHEL keep the commented vendor copy in `/usr/lib/systemd/resolved.conf` and leave `/etc/systemd/resolved.conf` as an empty placeholder, while Ubuntu 26.04 and CachyOS put the commented copy in `/etc/systemd/resolved.conf`. Open the empty file, or a path that does not exist on your system, and you see none of the shipped defaults. systemd recommends drop-ins for local changes, and recommends the 60 to 90 range in `/etc/` for them, hence the `90-` prefix. The bare `DNS=` handles the second trap: `DNS=` takes a list, and systemd collects list entries from every file it reads rather than letting the last one win. An empty assignment clears what was collected, leaving only the four Quad9 addresses; without it, a resolver configured earlier stays in the list. Empty `FallbackDNS=` works the same way.

Each line is written the way it is for a reason:

- `DNS=` lists all four addresses of the Quad9 service that does malware blocking and DNSSEC validation, the set Quad9 labels Recommended: the IPv4 pair `9.9.9.9` and `149.112.112.112`, and the IPv6 pair `2620:fe::fe` and `2620:fe::9`, so the setup works on a dual-stack network. They belong on this line rather than on `FallbackDNS=`, which the manual describes as "only used if no other DNS server information is known".
- `#dns.quad9.net` is the part that matters most for DoT. The `address#server_name` form tells `systemd-resolved` to validate the server's TLS certificate against that hostname and to send it as the SNI (Server Name Indication: the field that tells a TLS server which name you are asking for). Leave the suffix off and the certificate is checked against the server's IP address, with no SNI sent. Quad9's certificate lists its IP addresses as well as its names, so both forms validate, but the hostname form is the one Quad9 documents and it ties the connection to a name you chose. `dns.quad9.net` is the common name on the certificate its resolvers present on 853.
- `FallbackDNS=` is deliberately empty, though what that overrides depends on who built your systemd. Fedora builds systemd with no compiled-in fallback list at all. Arch-based builds, CachyOS among them, keep upstream's public fallback resolvers (Quad9, Cloudflare, and Google), each carrying a DoT hostname, so they follow your `DNSOverTLS=` setting rather than dropping to plaintext. To see which you have, look for a `Fallback DNS Servers:` line in `resolvectl status` before you make this change. Emptying the line is worth doing either way: it guarantees a resolver you did not choose is never consulted, even if your `DNS=` line fails to parse. The line is there for hardening, and it costs you failover: when Quad9 is unreachable, lookups fail instead of quietly going elsewhere.
- `DNSOverTLS=yes` is the strict mode, where lookups are encrypted or they fail. The alternative, `opportunistic`, drops to plaintext when the server appears not to support TLS, which an attacker can trigger by synthesizing a response; systemd's own manual warns that in that mode "the resolver is not capable of authenticating the server, so it is vulnerable to 'man-in-the-middle' attacks". Use `yes`.
- `DNSSEC=yes` turns on strict local validation, so a forged or badly-signed record produces a lookup failure rather than an answer. The manual recommends it wherever the DNS server is known to support DNSSEC correctly, which Quad9 does. The weaker `allow-downgrade` attempts validation and then switches it off if the server looks like it does not support DNSSEC, which the manual notes an attacker can also trigger deliberately. Where strict local validation causes trouble you cannot chase down, leaving this line at the default `no` is defensible: Quad9 validates upstream, and SP 800-81r3 accepts that a stub with a secured link to a validating resolver "can be considered to be 'using' DNSSEC". What you give up is seeing a failure yourself, because the upstream validator sees it and you get a general error back. Strict validation does occasionally break a real site whose zone is signed incorrectly, and it breaks captive portals; see Troubleshooting.

Both of these settings default to off, so each line is a real change

`DNSSEC=no AND DNSOverTLS=no ARE THE SHIPPED DEFAULTS`

`systemd-resolved` documents both `DNSSEC=` and `DNSOverTLS=` as defaulting to `no`, and Fedora, Ubuntu, and Arch all ship that default. An unconfigured machine therefore does no local DNSSEC validation and sends its lookups in the clear, whichever resolver it points at: changing only the resolver address changes who answers and leaves both properties as they were.

Apply it, then look at the merged result:

```
sudo systemctl restart systemd-resolved
systemd-analyze cat-config systemd/resolved.conf
```

The second command prints every file that contributes to the configuration, in the order systemd reads them, with your drop-in last. Use it to catch an uncommented `DNS=`, `DNSOverTLS=`, or `DNSSEC=` line left behind somewhere else.

04 Stop your router's DHCP-advertised DNS from being used as well

This step is easy to skip, and it is what keeps every lookup on the resolver you picked. The manual is explicit that requests "are sent to one of the listed DNS servers in parallel to suitable per-link DNS servers", and a normal Wi-Fi or Ethernet connection has per-link servers: the ones it learned over DHCP. NetworkManager pushes those into `systemd-resolved` through its D-Bus interface, so leaving them in place means a share of your lookups goes to your router.

- **What those router lookups do and do not inherit**
  A per-link server uses the link's own `DNSOverTLS` setting, and falls back to the global one when the link has none, which NetworkManager confirms: unset means "Systemd-resolved uses global setting." So the `DNSOverTLS=yes` from Step 3 covers those lookups too, so they fail against a router that does not offer DoT on 853. Where the link's server does speak TLS, the query still goes to a server you did not choose, carries no hostname for the certificate check, and skips Quad9's filtering entirely.

Tell NetworkManager to ignore the DNS servers DHCP advertises, so only your Quad9 configuration is left. Use one of the three methods below.

Option A: per connection with nmcli (recommended, and works on a headless server)

List the connections, then modify the one you use, replacing `<name>`:

```
nmcli connection show
sudo nmcli connection modify "<name>" \
  ipv4.ignore-auto-dns yes \
  ipv6.ignore-auto-dns yes
nmcli connection up "<name>"
```

- **Run the last command without sudo**
  On a desktop, the Wi-Fi password usually lives in your user's wallet (GNOME Keyring or KWallet), which root cannot read, so `sudo nmcli connection up` fails with "Secrets were required, but not provided". Running it as yourself lets the desktop agent supply the password. The alternatives are `--ask`, or toggling the connection off and on in the network applet. The `modify` command is saved either way, so any later reconnect picks it up.

With the global `DNS=` from Step 3 in place, this is enough on its own: the link now offers no DNS servers, so every query falls through to your Quad9 over DoT configuration.

Option B: pin Quad9 on the connection itself, and force DoT there too

Useful on a machine that sets no global `DNS=`. Include the `#dns.quad9.net` hostname so the certificate check still validates a name, and set DoT on the connection:

```
sudo nmcli connection modify "<name>" \
  ipv4.dns "9.9.9.9#dns.quad9.net,149.112.112.112#dns.quad9.net" \
  ipv6.dns "2620:fe::fe#dns.quad9.net,2620:fe::9#dns.quad9.net" \
  ipv4.ignore-auto-dns yes ipv6.ignore-auto-dns yes \
  connection.dns-over-tls yes
nmcli connection up "<name>"
```

Option C: the desktop network settings

- **GNOME (Ubuntu, Fedora Workstation):** open **Settings > Network** or **Wi-Fi**, click the gear beside your connection, and on the **IPv4** tab leave **Method** on *Automatic (DHCP)* while switching **DNS** from *Automatic* to **Off**. Repeat on **IPv6**, click **Apply**, and reconnect.
- **KDE (Fedora KDE, CachyOS):** in the network applet open **Configure Network Connections**, select your connection, and on the **IPv4** tab set **Method** to *Automatic (Only addresses)*, so DHCP still provides the address and stops providing DNS. Repeat on **IPv6**, apply, and reconnect.

The GUI cannot express the certificate hostname

A COMMON AND SILENT GAP

Both desktop editors accept plain IP addresses only, for example `9.9.9.9`. They have no field for the `#dns.quad9.net` suffix that drives TLS certificate-name validation and SNI, and they do not force strict DoT on the connection by themselves. Using the GUI means you are relying entirely on the global `DNSOverTLS=yes` and `DNS=` from Step 3 for the encryption and the certificate name, which is why switching DHCP-provided DNS **off** matters far more here than whichever addresses you type in. For per-connection certificate validation, use Option A or Option B.

05 Check that the configuration took effect

```
resolvectl status
```

The **Global** section should look close to this:

```
        Protocols: +LLMNR +mDNS +DNSOverTLS DNSSEC=yes/unsupported
  resolv.conf mode: stub
Current DNS Server: 9.9.9.9#dns.quad9.net
       DNS Servers: 9.9.9.9#dns.quad9.net 149.112.112.112#dns.quad9.net
                    2620:fe::fe#dns.quad9.net 2620:fe::9#dns.quad9.net
```

- **Read the DNSSEC status carefully: only the part before the slash is your setting**
  The DNSSEC state lives inside the `Protocols` line, and there is no separate `DNSSEC setting:` line to look for. `DNSSEC=yes/...` confirms strict validation is on. What follows the slash is `systemd-resolved`'s own probe of what the current server appears to support, it changes independently of your setting, and it can read `unsupported` while validation is working correctly. Treat Step 7 as the authoritative test and leave that token alone.

Your `LLMNR` and `mDNS` tokens may differ: they report two unrelated settings each distribution ships its own way (plus for enabled, minus for disabled, `LLMNR=resolve` for resolve-only), and neither concerns this guide. Column layout also shifts between systemd versions. Confirm `+DNSOverTLS`, `DNSSEC=yes` before the slash, `resolv.conf mode: stub`, and the four Quad9 addresses with their `#dns.quad9.net` suffixes. After Step 4 Option A, your active **Link** section lists no DNS servers of its own, so queries use the Global ones; that is the state you want.

06 Confirm the lookups are encrypted (853, and nothing on 53)

These checks use `tcpdump` from the cheat-sheet. Each capture runs until you press **Ctrl-C**, so you need two terminals: one capturing, one making a lookup. They capture on every interface (`-i any`); to watch a single one, find it with `ip -br link` and swap in `-i <iface>`. A warning that the `any` device does not support promiscuous mode is normal.

The negative check: no plaintext DNS should leave the machine

Traffic to the local stub on `127.0.0.53` and `127.0.0.54` is expected and healthy, so exclude all of loopback and watch what is left:

```
sudo tcpdump -n -i any 'port 53 and not net 127.0.0.0/8'
```

This should stay silent while you browse. Anything appearing here is a lookup leaving in the clear from something that is not using the stub. Running containers are the common case, for the reason given in the containers section near the end, and a browser with its own DoH resolver does the same. A per-link server that survived Step 4 will not show up here, because it inherits the global `DNSOverTLS=yes`; look for that one on 853 instead, going to an address that is not Quad9. Press **Ctrl-C** to stop.

The positive check: encrypted DNS to Quad9 on 853

Any of the four Quad9 addresses can be the active one, so match on the port rather than a single host. Start the capture in the first terminal:

```
sudo tcpdump -n -i any 'port 853'
```

In the second terminal, clear the cache so the lookup really travels upstream, then ask for something:

```
sudo resolvectl flush-caches
resolvectl query example.com
```

TLS traffic to a Quad9 address on 853, with the first capture still silent, is what confirms DoT is carrying your lookups. If you want to see the certificate for yourself, `openssl s_client -connect 9.9.9.9:853 -servername dns.quad9.net` prints the chain Quad9 presents; the subject common name is `dns.quad9.net`.

07 Confirm DNSSEC validation is actually rejecting bad records

Use the long-standing DNSSEC test pair: `sigok` is signed correctly and has to resolve, and `sigfail` is deliberately broken and has to be rejected. The `verteiltesysteme.net` names below are aliases (a CNAME chain) onto the current canonical `ippacket.stream` names, so either form works:

```
resolvectl query sigok.verteiltesysteme.net
resolvectl query sigfail.verteiltesysteme.net
```

- `sigok.verteiltesysteme.net` resolves, and the summary lines under the answer include `-- Data is authenticated: yes; Data was acquired via local or encrypted transport: yes`. The two flags on that line report different things: the first is DNSSEC, the second is DoT. Both should say yes here.
- `sigfail.verteiltesysteme.net` fails, with a message along the lines of `resolve call failed: DNSSEC validation failed: invalid`. The exact reason token varies between versions.

An optional cross-check. `dig` asks the same question a different way:

```
dig sigfail.verteiltesysteme.net    # expect status: SERVFAIL
dig sigok.verteiltesysteme.net      # expect an answer
```

- **The ad flag depends on what the query asked for, so read resolvectl instead**

  The `127.0.0.53` stub sets the `ad` (Authenticated Data) bit in its reply only when the query asked for it, by setting either the `ad` bit or the DNSSEC OK (DO) bit. `dig` sets `ad` by default, so it normally does see the flag; a client that sets neither gets `ad` cleared on data `systemd-resolved` did validate. The bit also says nothing about whether the transport was encrypted. `resolvectl query` reports both properties in words, so read that one.

Quad9's recommended service validates DNSSEC upstream as well, so with local validation on, a rejected `sigfail` means both resolvers turned it away and you saw the rejection yourself. With `DNSSEC=no` the pair still behaves the same way, because Quad9 refuses `sigfail` on its own; the test then tells you about Quad9 rather than about your host. Verified against Quad9 on 2026-08-08: `sigok` returns an answer with the `ad` flag set, `sigfail` returns SERVFAIL, and each name is an alias onto its counterpart under `rsa2048-sha256.ippacket.stream`.

LIMITS What this hides, and what it leaves in the open

Encrypted DNS hides the names you look up. Everything below stays visible anyway.

Destination IP addresses stay visible

Once the lookup is done, your device opens a connection to the site's address. Your ISP sees that address and can often map it back to a site, especially one on dedicated hosting.

The TLS handshake usually still announces the hostname

Most HTTPS connections send the site name in cleartext in the TLS ClientHello's SNI field. Encrypted Client Hello (ECH) fixes that, and it has to be in use at both ends, so treat the hostname as exposed unless you have confirmed otherwise.

Quad9 sees your queries

You have moved trust from your ISP and the local network to Quad9, a non-profit that states it does not log personal data and that disables ECS on this service. That is an improvement, though it still leaves one party able to see every name you ask for.

This is not anonymity

Nothing here changes where your traffic appears to come from. For that you need a VPN or Tor, a different tool with different trade-offs. See our WireGuard guide for the self-hosted version.

Applications can go around the system resolver

A browser with its own DoH setting resolves names itself and never consults `systemd-resolved`, so your careful configuration does not apply to it. SP 800-81r3 draws the same line, separating the OS-layer stub resolver from an application-layer stub that "operates independently within a specific application and bypasses the OS resolver". Check the DNS setting in each browser you use, and either turn its resolver off or point it at Quad9 too.

ALIGN How this maps to NIST and CIS

No CIS Benchmark covers a DNS client configuration. The Linux benchmarks reach DNS from the server side, checking that a host is not running a DNS server or dnsmasq it does not need, so the alignment for this work sits at the controls level:

WHAT THIS SETUP PROVIDES	NIST SP 800-53 REV 5	CIS CONTROLS V8.1
DNSSEC validation of every response, so a forged record never resolves	SC-21	-
Lookups encrypted between the stub resolver and Quad9	SC-8, SC-8(1)	-
Known-malicious domains blocked at resolution time	-	Safeguard 9.2 (IG1)

SC-21 asks you to "request and perform data origin authentication and data integrity verification on the name/address resolution responses the system receives from authoritative sources", which is what `DNSSEC=yes` does on this host. CIS Safeguard 9.2 asks you to "use DNS filtering services on all end-user devices, including remote and on-premises assets, to block access to known malicious domains"; it sits in Implementation Group 1, so it applies to the smallest organizations, and pointing a device at Quad9's blocking service is the measure it describes. Listing all four Quad9 addresses is ordinary resilience, and neither catalogue treats it as a control: SC-22 governs the name resolution service an organization runs itself, and says nothing about which resolver a client points at.

Cite the current NIST DNS guide, because the old one was withdrawn this year

SP 800-81 REV 3 SUPERSEDES SP 800-81-2

NIST SP 800-81 Rev. 3, "Secure Domain Name System (DNS) Deployment Guide" (March 2026), is the current deployment guidance, and it covers DNSSEC and encrypted DNS together. It replaced SP 800-81-2, which NIST withdrew on 19 March 2026. SP 800-53 Rev. 5 still lists the withdrawn SP 800-81-2 under SC-21's references, since Rev. 5 predates the change, so quote Rev. 3 when you write this up. Its Section 4.2.2 asks organizations to "restrict stub resolvers to only use encrypted DNS on authorized DNS services wherever possible". Note the word authorized: the same section also tells organizations to block unauthorized outbound DoT on 853, so on a managed network this setup belongs to whoever decides which resolver is approved.

Regulated environments: check the cryptography requirement before you deploy this widely

FIPS AND CIPHER POLICY

SP 800-81r3 notes that some organizations require FIPS-140-approved cryptographic modules for TLS, which would extend to encrypted DNS, and it requires name servers, where they support it, to allow only the ciphers permitted in SP 800-52. DoT runs over ordinary TLS, so on RHEL the system-wide cryptographic policy governs the handshake and can be set to `FIPS`. Where you depend on that, confirm your policy with `update-crypto-policies --show` and satisfy yourself that the resolver you chose negotiates within it.

HELP Troubleshooting

- **One specific site stopped resolving**

  Its zone may be signed incorrectly, or you may be behind a captive portal (hotel or airport Wi-Fi) whose sign-in page hijacks DNS. Captive portals break strict DoT and DNSSEC routinely, and because `DNSOverTLS=yes` with an empty `FallbackDNS=` fails closed, the portal's own login page may not resolve either. To get online, set the drop-in aside rather than editing it, since systemd reads only `*.conf` in that directory: rename it to `90-quad9-dot.disabled`, run `sudo systemctl restart systemd-resolved`, sign in, then rename it back and restart again. Set yourself a reminder, because nothing warns you if the relaxed state stays.

- **All DNS resolution stopped**

  With `FallbackDNS=` empty, an unreachable Quad9 means no resolution, by design. Test the path with `nc -vz 9.9.9.9 853`. A refused or timed-out connection points at a network that blocks outbound 853, which some guest and corporate networks do; a working connection points at the local configuration instead, so re-read Step 3 for a typo.

- **Lookups still appear on port 53 to an external server**

  Either something is resolving outside the stub, or a link has DoT switched off. Run `resolvectl status` and look for a Link section that lists a DNS server of its own or shows `-DNSOverTLS`, then re-check Step 4 there (`ignore-auto-dns`, or the GUI DNS switch set to **Off**). A VPN that came up after your change can reintroduce a link server. If every Link section is clean, the traffic belongs to an application or container that bypasses `systemd-resolved` entirely.

- **resolvectl shows DNSOverTLS off on a Link**

  Something else owns that link's DNS: a VPN client, or CachyOS-Welcome's `blocky` DoH or DoQ option. Pick one resolver for the machine and disable the others, because two of them will keep contradicting each other.

- **(RHEL) DoT will not connect even though Quad9 is reachable**

  The system-wide cryptographic policy governs the TLS handshake. The `DEFAULT` policy works with Quad9; a hardened custom policy, or `FUTURE`, can restrict the ciphers or protocol versions the handshake needs. Check with `update-crypto-policies --show`.

- **(CachyOS) DNS works only after you restart systemd-resolved**

  On some setups resolution fails at boot until the service settles. Confirm it is enabled with `systemctl is-enabled systemd-resolved`, and check the ordering with `systemctl status systemd-resolved NetworkManager`.

- **Everything resolves, but Data is authenticated says no**

  Most zones are still unsigned, and an unsigned zone cannot be authenticated, so `no` on an arbitrary domain is normal. Test with `sigok.verteiltesysteme.net` from Step 7, a signed zone, instead of a domain you happen to use.

VPN VPNs, Tailscale, and containers

Tailscale

It installs its own resolver at `100.100.100.100` and, with the admin console's "Override DNS servers" option enabled, takes every query ahead of your Quad9 configuration while the VPN is up. To keep Quad9 over DoT in effect, either turn that override off or set Quad9 as the global nameserver in Tailscale's own DNS settings.

Other VPNs

Most push their own resolver for the life of the connection, which is usually what you want, because it keeps the lookups inside the tunnel where the local network cannot see them. Our WireGuard guide sets an in-tunnel resolver for that reason.

Containers reach Quad9 on plaintext 53, outside your DoT connection

A container has its own network namespace, so `127.0.0.53` is the container's own loopback rather than the host's stub. Docker works around that by building the container's `/etc/resolv.conf` from `/run/systemd/resolve/resolv.conf`, where `systemd-resolved` lists the real upstream servers; it does so when `127.0.0.53` is the only nameserver on the host. The stub symlink puts it there. After Step 3 those upstreams are the Quad9 addresses, so container lookups do reach Quad9 and get its filtering and upstream DNSSEC, on plaintext port 53 with no local validation. That is the usual reason the Step 6 negative capture goes noisy. Where Docker cannot trace a loopback resolver back to upstreams it substitutes public servers instead, logging "Using default external servers : [8.8.8.8 8.8.4.4]". To pin the addresses either way, put `"dns": ["9.9.9.9", "149.112.112.112"]` in `/etc/docker/daemon.json` and restart Docker, or pass `--dns` per container, which Podman accepts too.

ROLLBACK How to roll back

```
sudo rm /etc/systemd/resolved.conf.d/90-quad9-dot.conf
sudo systemctl restart systemd-resolved

# only if you changed a NetworkManager connection in Step 4 (Option A or B)
sudo nmcli connection modify "<name>" ipv4.ignore-auto-dns no ipv6.ignore-auto-dns no \
  ipv4.dns "" ipv6.dns "" connection.dns-over-tls default
nmcli connection up "<name>"
```

Compare `resolvectl status` against the `~/dns-before.txt` you saved in Step 2 to confirm you are back where you started. If you used a desktop GUI in Option C, reopen the connection editor, set IPv4 and IPv6 DNS back to *Automatic*, clear any addresses you typed, then apply and reconnect.

To undo Step 1 on RHEL as well: remove `/etc/NetworkManager/conf.d/10-resolved.conf`, delete the stub symlink with `sudo rm /etc/resolv.conf` (NetworkManager will not overwrite a symlinked `resolv.conf`), run `sudo systemctl disable --now systemd-resolved`, then `sudo systemctl restart NetworkManager` to have it write a fresh managed `/etc/resolv.conf`.

DONE What you achieved

- Lookups go to Quad9's malware-blocking service on all four addresses, IPv4 and IPv6, inside TLS on port 853, authenticated against the `dns.quad9.net` certificate and failing rather than dropping to plaintext.
- Responses are DNSSEC-validated on this host by `DNSSEC=yes`, and again upstream by Quad9, so a forged record comes back as an error.
- Your router's DHCP-advertised resolver is suppressed, and Quad9 sends no EDNS Client Subnet data, so authoritative servers do not learn your approximate location.
- You can prove all of it: port 853 busy and port 53 silent in Step 6, and `sigfail` rejected in Step 7.

REF References

Quad9: Service Addresses & Features · `quad9.net/service/service-addresses-and-features`

systemd `resolved.conf(5)` (the directives, and drop-in precedence) · `freedesktop.org/software/systemd/man/latest/resolved.conf.html`

NetworkManager `nm-settings-nmcli(5)` (`ignore-auto-dns`, `connection.dns-over-tls`) · `networkmanager.dev/docs/api/latest/nm-settings-nmcli.html`

Red Hat: Setting up systemd-resolved, and the release notes for your RHEL minor version for its support status · `docs.redhat.com/en/documentation/red_hat_enterprise_linux`

Docker: container DNS and unusable loopback resolvers · `docs.docker.com/engine/daemon/troubleshoot`

NIST SP 800-81 Rev. 3, Secure Domain Name System (DNS) Deployment Guide (March 2026) · `csrc.nist.gov/pubs/sp/800/81/r3/final`

NIST SP 800-53 Rev. 5 (SC-8, SC-21, SC-22) · `csrc.nist.gov/pubs/sp/800/53/r5/upd1/final`

CIS Critical Security Controls v8.1, Safeguard 9.2 · `cisecurity.org/controls`

ArchWiki: systemd-resolved · `wiki.archlinux.org/title/Systemd-resolved`