VeilDNS: Post-Quantum DNS Privacy from the First UDP Packet

Aditya Singh Rawat
Ashoka University
aditya.rawat@alumni.ashoka.edu.in

Mahabir Prasad Jhanwar
Ashoka University
mahavir.jhawar@ashoka.edu.in

Rei Safavi-Naini
University of Calgary
rei@ucalgary.ca

***Abstract*—DNS was designed around UDP, and most DNS traffic today still follows that design: one query, one response, and no handshake. This makes DNS fast and scalable, but exposes client queries in cleartext. DoT, DoH, and DoQ provide privacy by encrypting DNS messages with TLS. This works once TLS/TCP or QUIC handshakes complete, but cold sessions spend extra round trips and CPU before the first private query can be sent—a cost that grows with post-quantum cryptography.**

We present VeilDNS, a DNS-native protocol that uses a post-quantum KEM and AEAD to encrypt DNS questions and answers and authenticate the resolver in a single UDP exchange. Given an authenticated resolver KEM key, clients have query privacy from their first UDP packet. VeilDNS supports stateless and stateful modes—the latter reducing recurring computation and bandwidth. An optional mode provides forward secrecy from the second query onward. By preserving the DNS wire format, VeilDNS runs on the standard UDP port 53.

Compared with ML-DSA-authenticated Do(T/H/Q) on the well-provisioned network profile, VeilDNS's stateless and stateful modes achieve up to $3.5\times$ higher cold-session throughput, $4.8\times$ lower average latency, $4.7\times$ lower CPU time, $13.7\times$ fewer bytes, and $15.8\times$ fewer packets. On the constrained profile, the throughput gain reaches $12.4\times$. Tamarin establishes query confidentiality, response authentication, and forward secrecy.

I. Introduction

The Domain Name System (DNS) [1] commonly maps a domain name (`foo.bar`) to an IP address (`1.2.3.4`). It is a query-response protocol: a client sends a query to a resolver and receives an answer, usually in one UDP exchange. UDP is the default transport because it avoids connection setup, keeping lookups fast and scalable. TCP remains available for larger[1] responses that would otherwise fragment over UDP/IP and create reliability and security risks [2], [3], [4], [5].

DNS was originally designed for public lookup, not for secure client-to-resolver communication [6]. Queries and responses normally travel in cleartext, allowing eavesdroppers to learn the domains a client queries; without resolver authentication and message-integrity checks, an on-path[2] attacker can also spoof or tamper with responses. DNS-over-TLS (DoT), DNS-over-HTTPS (DoH), and DNS-over-QUIC (DoQ), collectively DoX,[3] address this by carrying DNS messages inside authenticated and encrypted channels [9], [10], [11]. These channels work well for long-lived sessions, but the first private query must wait for channel establishment and resolver authentication—producing a traffic shape far removed from the stateless UDP exchange of classic DNS.

[1]Responses exceeding the path MTU, typically 1500 B.

[2]Off-path spoofing is also possible but requires guessing the UDP source port, DNS transaction `ID`, and any Client Cookie [7], [8].

[3]We use DoX henceforth as shorthand for DoT, DoH, and DoQ.

TABLE I
Signature comparison. Adapted from Table VII. Sizes are in bytes; multipliers are relative to ECDSA.

Signature	Public Key	Signature	Sign cost
ECDSA P-256	64 (1×)	64 (1×)	1×
Falcon-512	897 (14×)	752 (12×)	13×
ML-DSA-44	1312 (21×)	2420 (38×)	5×
SLH-DSA-SHA2-128s	32 (0.5×)	7856 (123×)	22k×

Cold start is where post-quantum cryptography (PQC) magnifies the connection cost. PQC key-exchange and signature artifacts are much larger than their classical counterparts, directly increasing bandwidth use. For cold DoX, online resolver signing is a major cryptographic server-side cost (Table I): the resolver signs each fresh TLS or QUIC handshake. Long-lived DoX channels can amortize this handshake cost over later DNS queries [12], [13], [14]. However, warm state remains fragile in practice. Prominent public resolvers close idle DoX connections within minutes, and sometimes after only 10 s (Table III). Mobile sleep/wake cycles, network roaming, NAT/firewall expiry, and connectivity drops can also force a fresh connection [9], [11], [15], [16].

VeilDNS achieves the client-query privacy and resolver authentication goals of DoX while retaining DNS's UDP query-response model. A transaction behaves like ordinary DNS over UDP/53: it completes in one UDP exchange, requires no per-session handshake, and can run statelessly.

Specifically, a client with a trusted resolver KEM public key sends a DNS-formatted query with a *dummy* public question. The *real* question is AEAD-encrypted under a key derived from the resulting KEM shared secret and carried in an extra DNS record alongside any KEM material needed for that mode. The resolver derives the same symmetric key, decrypts the real question, and returns a *dummy* DNS response carrying the encrypted *real* answer in an extra DNS record. A successful client-side decryption also authenticates the resolver.

VeilDNS requires the client to obtain and authenticate the resolver's KEM public key before use. This can be done

through a certificate [17], a DNSSEC-protected record [18], [19], or preconfiguration; key distribution itself remains outside the protocol. Once validated, the resolver KEM key can remain cached by the client and reused for later queries.

VeilDNS runs in three modes: stateless, stateful, and stateful-FS (forward secret). In **stateless** mode, the client performs a fresh KEM encapsulation for each query and the resolver keeps no session state, preserving the design shape of classic DNS/UDP. Here, every query carries a fresh KEM ciphertext. **Stateful** mode performs the KEM exchange once, caches the derived symmetric keys, and omits recurring KEM ciphertexts in later queries. **Stateful-FS** uses two UDP exchanges to establish forward-secret state. The first exchange matches stateful mode, except that the response also carries a fresh ephemeral KEM public key from the resolver, authenticated as part of that response. This key enables the second and later exchanges to be forward-secret. A future compromise of the resolver's static KEM key does not reveal queries sent from the second exchange onward.

Our contributions are:

- We design VeilDNS, a client-query privacy and resolver authentication protocol using a post-quantum KEM that encrypts DNS questions and answers inside DNS/UDP packets while preserving the standard DNS wire format.
- We define three operating modes: stateless, stateful, and stateful-FS. Stateful mode caches derived keys, trading resolver statelessness for lower recurring byte and CPU costs; stateful-FS makes the second and later queries forward-secret against future static-key compromise.
- We model VeilDNS in Tamarin and prove query and accepted-answer secrecy, response authentication, resolver-to-client master-key secrecy, and packet-key non-reuse across all modes; replay rejection under the stated replay assumption for established stateful traffic; and forward secrecy for stateful-FS mode.
- We implement all VeilDNS modes and evaluate them against vanilla DNS and post-quantum DoX baselines over local and cloud paths, reporting latency, throughput, IP-layer byte counts, packet counts, and CPU time.

VeilDNS combines direct UDP privacy with the standard DNS wire format. This matters in practice: prior access-network and cellular measurements identified paths where DNS-aware middleboxes rejected non-DNS payloads sent over UDP/53 [20], [21]. Because VeilDNS packets remain well-formed DNS messages, DNS tooling and DNS-aware middleboxes can parse the outer DNS headers, sections, and records [22]. Table II places the overall design in context by comparing VeilDNS with vanilla DNS and DoX.

TABLE II
CLIENT-TO-RESOLVER DNS PROTOCOL COMPARISON. FOR VEILDNS, † DENOTES STATELESS MODE AND ‡ DENOTES STATEFUL-FS.

Property	Vanilla	VeilDNS	DoX
Transport	UDP	UDP	TCP or QUIC
Port	53	53	853 or 443
Per-session handshake	No	No	Yes
1st-query privacy	No	Yes	Post-handshake
Resolver authentication	No	Yes	Yes
Stateless	Yes	Yes†	No
Forward secrecy	No	From 2nd query‡	Yes

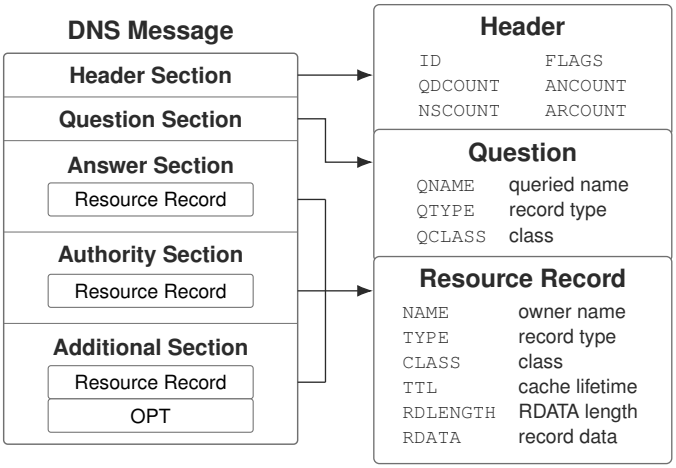


Fig. 1. **DNS message format.**

II. PRELIMINARIES

Notation. A DNS resource record (RR) is referred to simply as a record. ‖ denotes concatenation. For presentation, we often omit the root label (*i.e.*, the trailing period `(.)` as in `foo.bar.`) while writing fully qualified domain names. For DNS records, $X \to Y$ denotes field Y of record X. We collectively refer to stateful and stateful-FS as stateful(-FS), and to stateful and stateless as stateful/less.

A. DNS

In a typical DNS resolution, a client application issues a lookup through the local stub resolver, which sends a DNS query to a recursive resolver that answers from cache when possible. Otherwise, the recursive resolver iteratively queries authoritative name servers (ANSes), which hold the DNS records for their zones. The recursive resolver then returns the final answer to the stub resolver [23].

A DNS message consists of five sections (Fig. 1). The 12-byte Header section contains a 16-bit transaction `ID`, used to match responses to queries; a 16-bit `FLAGS` field; and 16-bit `QDCOUNT`, `ANCOUNT`, `NSCOUNT`, and `ARCOUNT`, which list the number of questions or records in the Question, Answer, Authority, and Additional sections, respectively. The Question section carries `QNAME`, `QTYPE`, and `QCLASS` (typically `IN` for Internet); the Answer, Authority, and Additional sections carry records with the following fields: `NAME`, `TYPE`, `CLASS` (usually `IN`), `TTL`, `RDLENGTH`, and `RDATA` [24].

On the wire, DNS names are encoded as a sequence of length-prefixed labels ending in the zero-length root label; *e.g.,* `foo.bar` is encoded as `[3]foo[3]bar[0]`. `A` and `AAAA` records map domain names to IPv4 and IPv6 addresses, respectively; `NS` records specify domain names of ANSes; `TXT` records carry arbitrary strings.

TABLE III
DoX idle timeouts. After one successful DNS query from our network on July 18, 2026, we left each connection idle and recorded its closure; N/A denotes no public DoQ service.

Resolver	DoT	DoH	DoQ
Google	1 min	4 min	N/A
Cloudflare	10 s	6 min 40 s	N/A
Quad9	11 s	3 min	3 min
AdGuard	30 s	2 min	5 min

DNS Cookies. EDNS(0) [25] extends DNS with an `OPT` pseudo-record in the Additional section, where various options are encoded. The DNS Cookie [8] option (code `10`) carries an 8-byte Client Cookie in the query and, after the server responds, an 8–32-byte Server Cookie. The cookies are derived from endpoint secrets and the client and server IP addresses.

The client's initial query carries the Client Cookie, which the server echoes alongside a Server Cookie in the response. Echoing the Client Cookie adds entropy to response validation, helping the client reject off-path forgeries. In later queries, the client includes both cookies, allowing the server to validate source-address reachability without keeping per-client state. This mitigates spoofed-query abuse, including server resource exhaustion and DDoS amplification and reflection.

Size constraints on DNS/UDP. EDNS(0) lets endpoints advertise their DNS message size limits: `OPT` → `CLASS` (the `CLASS` field of `OPT`) carries the requester's receive limit in a query and the responder's receive limit in a response. Although EDNS(0) allows messages approaching 64 KB, implementations generally support much smaller buffers, commonly no more than 4096 bytes. In practice, DNS/UDP messages are commonly kept within 1232 bytes to avoid IP fragmentation; this corresponds to the IPv6 minimum MTU of 1280 bytes minus the 40-byte IPv6 and 8-byte UDP headers [4], [5].

B. DoT, DoH, and DoQ

Classic DNS exposes the entire message to a network observer between client and resolver. DoX protocols protect this hop by carrying DNS messages inside secure transports.

DoT runs DNS over TLS/TCP; DoH carries DNS messages as HTTPS request/response bodies; and DoQ carries DNS over QUIC streams. DoT uses a TCP byte stream, DoH multiplexes HTTP/2 streams over TCP, and DoQ assigns queries to independent QUIC streams, avoiding TCP head-of-line blocking. Once established, these channels can carry recurring DNS messages efficiently, but a timeout or network interruption often requires a new connection; without usable resumption state, this entails a full handshake. Moreover, public-resolver idle timeouts are not especially generous (Table III).

DoX is not a replacement for DNSSEC [26]. DoX protects client privacy via an encrypted and authenticated channel between the client and resolver. DNSSEC operates between the resolver and ANSes and provides authenticity and integrity, but not confidentiality, for DNS records through a chain of trust anchored in the DNS root. Thus, DoX provides *path* security while DNSSEC provides *object* security.

C. Cryptographic Primitives

A key encapsulation mechanism (KEM) defines three operations: KeyGen creates a public/private key pair; $\mathsf{Encap}(\mathsf{pk})$ returns a ciphertext ct and a shared secret ss; and $\mathsf{Decap}(\mathsf{sk}, \mathsf{ct})$ recovers ss. A key derivation function (KDF) derives symmetric keys from shared secrets and additional inputs. Authenticated encryption with associated data (AEAD) encrypts a plaintext under a key and nonce, producing a ciphertext and authentication tag. The tag authenticates the ciphertext and the associated data (AEAD-AD), which remains unencrypted.

D. Terminology

We use three operational phases throughout the paper:

- **Setup phase:** endpoints provision and authenticate resolver keys, certificates, or pinned configuration; this phase is not measured during experiments.
- **Cold phase:** endpoints begin with no session state. Stateful and stateful-FS cache master keys in one and two UDP exchanges, respectively; DoX establishes a connection.
- **Warm phase:** endpoints begin with session state. Stateful(-FS) uses cached master keys; DoX uses existing channels.

Vanilla DNS and stateless mode do not create session state.

III. VeilDNS

VeilDNS is a wire-compatible DNS/UDP protocol that provides client-query privacy and resolver authentication in one DNS query-response exchange. It combines a post-quantum KEM with AEAD encryption: once setup gives the client an authenticated resolver KEM public key, the client can send its first DNS question with 0-RTT privacy. The resolver decapsulates, decrypts, and returns an AEAD-protected DNS answer; successful client-side decryption provides unilateral key confirmation and authenticates the resolver as the holder of the corresponding KEM private key [27].

VeilDNS preserves the standard DNS wire format. The visible `QNAME` is a dummy domain such as `veildns.invalid`, under the reserved `.invalid` TLD [28]; the real question is AEAD-encrypted under KEM-derived key material and carried in a `TXT` record in the Additional section. The resolver decrypts the real question, resolves it, encrypts the real answer in a `TXT` record, and returns it in the DNS response.

VeilDNS supports three modes: stateless, stateful, and stateful-FS. **Stateless** mode keeps no session state: the client performs a fresh KEM encapsulation for each query, and every query carries a KEM ciphertext. **Stateful** mode performs the KEM exchange once, caches the derived master keys, and omits recurring KEM ciphertexts on later queries. **Stateful-FS** uses two UDP exchanges. The first response carries an ephemeral resolver KEM public key, to which the client encapsulates in the second exchange. The second exchange and later queries use forward-secret master keys derived from both KEM shared secrets. Fig. 2 shows the VeilDNS protocol flow at an abstract level.

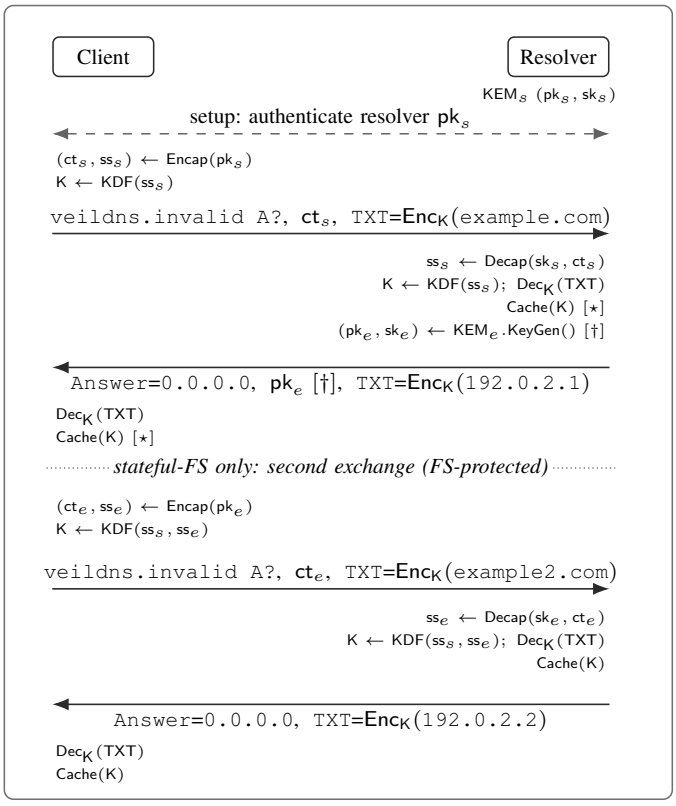


Fig. 2. **VeilDNS protocol.** $\star/\dagger$: stateful/stateful-FS; s/e: static/ephemeral KEM values. K denotes a directional master key. Enc_K and Dec_K derive the one-time key and nonce before performing AEAD. For clarity, the KDF shows only KEM shared-secret inputs; Algorithm 1 gives the complete flow.

A. Setup phase

VeilDNS setup consists of obtaining and authenticating the resolver's static KEM public key. The protocol remains agnostic to the key validation path: clients can authenticate the resolver KEM key through TLS PKI (*e.g.*, a KEMTLS certificate [17]), DNSSEC PKI (*e.g.*, a `TLSA` record [18] or a `DNSKEY` record [19]), or a local trust policy.

After validation, the client caches the resolver KEM key and reuses it for later queries. The cache lifetime is bounded by the certificate validity period, the validated DNS record's cache lifetime, or local policy, depending on the validation path. Setup also gives the client an authenticated list of the resolver's supported modes and cryptographic suites. Sections VI-C and VI-D give further details.

The protocol derives the resolver KEM key identifier as the low-order 16 bits of SHA-256 applied to the raw public key bytes, rather than from certificate metadata or DNSSEC `KeyTag`. The resulting identifier selects the matching resolver private key while remaining independent of how the public key was authenticated during setup. Although key-identifier collisions are unlikely with the small number of concurrently active keys, a resolver must check for duplicate identifiers within each algorithm suite before advertising a key.

B. Wire format

VeilDNS messages use standard DNS wire format over UDP/53. The visible question in queries is a dummy value:

`veildns.invalid.`	`A`	`IN`
QNAME	QTYPE	QCLASS

The real DNS question is encoded and padded[4] to a configured size as:

$$\texttt{QNAME} \,\|\, \texttt{QTYPE} \,\|\, \texttt{QCLASS} \,\|\, \texttt{padding}.$$

AEAD encrypts this plaintext, and the resulting ciphertext and authentication tag are carried in the `RDATA` field of a `TXT` record[5] as:

$$\texttt{context} \,\|\, \texttt{tag} \,\|\, \texttt{ciphertext}.$$

The 256-bit `context` is a public per-packet value used to derive unique AEAD key and nonce material; `tag` is the 128-bit AEAD authentication tag. Stateless queries and responses sample fresh random contexts. For stateful(-FS), the context is `sid` ‖ `"VEILDNS0"` ‖ `seq` on the client side and `sid` ‖ `"VEILDNS1"` ‖ `seq` on the resolver side, where `sid` is the 128-bit session identifier used to distinguish sessions and locate their cached keys and replay state, and `seq` is the 64-bit sequence number used for replay detection (§ A-C1). For the first stateful(-FS) exchange, the client uses a fresh sid_0 in its query, and the resolver returns a fresh sid_1 in its response. Both endpoints then use sid_1 for all later queries and responses. In all modes, `context` is included in the AEAD-AD.

VeilDNS first derives directional master keys from the KEM secret(s). For each packet, it then derives a one-time AEAD key and nonce from the corresponding master key and packet context. The master key is never directly used for AEAD. Together with fresh or sequence-based packet contexts, this gives each fresh encryption distinct key-and-nonce material.

In both directions, the AEAD-AD includes the visible DNS transaction `ID` and the DNS header `FLAGS`. For queries, this binds fields such as `OPCODE` (operation code). For responses, it binds fields such as `RCODE` (return code).

KEM public keys and ciphertexts are carried in private-use record type `65280` [29], denoted `PRIVATE` henceforth. Its `RDATA` contains the resolver KEM key identifier, mode, and cryptographic suite, followed by a typed KEM public key or ciphertext (Table IV). These fields are cryptographically bound through the KDF inputs and AEAD-AD (§ A-B).

Responses include a 0-`TTL` `A` record[6] containing a dummy IP in the Answer section:

`veildns.invalid.`	`A`	`IN`	`0`	`0.0.0.0`
NAME	TYPE	CLASS	TTL	RDATA

The real answer that is padded and encrypted is:

$$\texttt{ANCOUNT} \,\|\, \texttt{NSCOUNT} \,\|\, \texttt{ARCOUNT} \,\|\, \texttt{RR}_{1..n} \,\|\, \texttt{padding}.$$

[4]Padding reduces leakage of plaintext size through ciphertext length.
[5]Encoded as one or more length-prefixed character-strings [24].
[6]`TTL` is set to 0 to prevent middleboxes from caching the dummy record.

TABLE IV
PRIVATE→RDATA IN VEILDNS.

Field	VeilDNS use
`key ID`	resolver KEM key identifier (16 bits)
`mode`	`0x01` stateless, `0x02` stateful, `0x03` stateful-FS
`suite`	`0x01`: ML-KEM-512, HKDF-SHA256, AES-256-GCM
`payload`	`0x01` ‖ KEM ciphertext or `0x02` ‖ KEM public key

$\texttt{RR}_{1..n}$ is the ordered wire encoding of records in the Answer, Authority, and Additional sections, excluding the trailing `OPT`.[7] The receiver reconstructs the original header and question before parsing these records, preserving their original positions and DNS compression pointers. The resulting `context`, `tag`, and `ciphertext` are carried in `TXT` → `RDATA` as described earlier.

Every VeilDNS packet also carries experimental-use EDNS(0) option `65001` with value `0x01` for VeilDNS version 1; endpoints validate this option before processing any VeilDNS-specific records. A copy of the `OPT` record, with its `CLASS` field set to zero, is included in the AEAD-AD. This authenticates the option list and EDNS(0) flags while allowing middleboxes to rewrite the advertised UDP payload size.

C. Stateless mode

Fig. 3 shows the cold stateless wire format. Assume setup has given the client a trusted resolver KEM public key pk_s. The client hashes the raw pk_s and takes the low-order 16 bits as the key identifier kid_{pk_s}; here, $\mathsf{kid}_{pk_s} = \texttt{0x7a31}$. To query[8] `example.com A`, the client encapsulates to pk_s, obtaining KEM ciphertext ct_s and shared secret ss_s, derives directional master keys from ss_s, and encrypts the real question.

From ss_s, both sides derive separate master keys for client-to-resolver traffic (K_{c2s}) and resolver-to-client traffic (K_{s2c}), using the respective KDF direction labels `"c2s"` and `"s2c"`. The derivation is also bound to kid_{pk_s} and ct_s, as well as the mode and algorithm suite. For each packet, the endpoints derive its AEAD key and nonce from the directional master key and a fresh random context. The AEAD-AD includes the DNS transaction `ID`, packet context, and DNS header `FLAGS`, together with the mode and algorithm suite.

The query uses the dummy question. Its Additional section carries a `PRIVATE` record with `key ID=0x7a31`, `mode=0x01`, `suite=0x01`, and `payload=0x01 ||` ct_s. It also carries a `TXT` record with the encrypted `example.com A IN` question. The resolver selects its matching KEM private key using the client-sent kid_{pk_s}, decapsulates ct_s, derives the same directional master keys, and decrypts the real question.

The response repeats the same dummy question. Its visible answer contains only the dummy `A` record with `TTL` 0, while the real `example.com A` answer is encrypted in `TXT`→`RDATA`. For each response encryption, the resolver samples a fresh 256-bit packet context independently of the query context. Thus, even if an attacker replays the same stateless query, the resolver derives a distinct one-time packet key without maintaining a replay cache. Since this mode is stateless, neither side caches KEM-derived master keys across queries; each query carries a fresh KEM ciphertext.

D. Stateful mode

The stateful mode uses the same one-exchange shape as stateless. The client sends a fresh KEM ciphertext and a fresh sid_0 with an encrypted real question. The resolver decapsulates, decrypts the real question, and returns the encrypted real answer with a fresh sid_1. The initial master-key derivation binds sid_0; the session master-key derivation binds both sid_0 and sid_1. Both endpoints cache the session master keys and replay state under sid_1. To limit state created by spoofed first-contact queries, the resolver initially marks the cached state as pending; a later query with a valid Server Cookie and AEAD tag confirms the state allocation (§ A-C3).

On later queries, the client omits the `PRIVATE` KEM record and uses sid_1 in the packet context of the encrypted `TXT` → `RDATA` payload. The resolver uses sid_1 to find the cached state, checks `seq` for freshness, and derives the query's packet key from its cached K_{c2s} for decryption. It then derives a packet key from its cached K_{s2c} using the same `seq` and uses it to encrypt the response. The client likewise checks this `seq` for freshness before accepting the response.

E. Stateful-FS mode

The stateful-FS mode extends the stateful mode with a second DNS exchange that is already forward-secret. The first stateful-FS exchange is the same as in stateful mode, except that the resolver's response also carries a fresh ephemeral KEM public key pk_e. On the wire, it is carried in `PRIVATE` → `RDATA` as `0x02` ‖ pk_e. This key is included in the AEAD-AD for the encrypted first answer. The client accepts pk_e only if that answer verifies under the static first-exchange keys.

In the second exchange, the client encapsulates to pk_e, derives provisional forward-secret keys from ss_s and ss_e, and sends the KEM ciphertext ct_e with the encrypted real question. The resolver decapsulates ct_e, derives the same provisional FS keys, decrypts the real question, encrypts the real answer, and sends the DNS response. The provisional keys protect only the second exchange. After that exchange succeeds, both endpoints derive final FS master keys from ss_s and ss_e and cache them under sid_1. The sequence counter continues across all key phases. Later queries have the same visible shape as stateful queries, but use the cached FS master keys.

F. Protocol summary

§ A-B summarizes the key-derivation and AEAD-AD inputs for VeilDNS. Algorithm 1 describes the online operations, and Table V compares the modes.

[7] Our prototype accepts exactly one `OPT` record positioned last in the Additional section and rejects other placements or multiplicities.

[8] This procedure applies to other query types (e.g., `AAAA`, `NS`, and `TXT`).

VeilDNS query for `example.com A`					
Header					
ID	FLAGS	QDCOUNT=1	ANCOUNT=0	NSCOUNT=0	ARCOUNT=3
Question (dummy)					
QNAME=veildns.invalid.		QTYPE=A		QCLASS=IN	
Additional: PRIVATE record (KEM ciphertext)					
NAME=veildns.invalid.		TYPE=PRIVATE	CLASS=IN	TTL=0	RDLENGTH
RDATA: key ID=0x7a31 || mode=0x01 || suite=0x01 || payload=0x01 || ct_s					
Additional: TXT record (encrypted real question)					
NAME=veildns.invalid.		TYPE=TXT	CLASS=IN	TTL=0	RDLENGTH
RDATA: context=random256 || tag || AEAD.Enc(example.com || A || IN || padding)					
Additional: OPT					
NAME=.		TYPE=OPT	CLASS=1232	TTL=0	RDLENGTH
RDATA: Code=10 || len=8 || ClientCookie Code=65001 || len=1 || 0x01 (VeilDNS version)					

VeilDNS response for `example.com A`					
Header					
ID	FLAGS	QDCOUNT=1	ANCOUNT=1	NSCOUNT=0	ARCOUNT=2
Question (dummy)					
QNAME=veildns.invalid.		QTYPE=A		QCLASS=IN	
Answer: A record (dummy)					
NAME=veildns.invalid.		TYPE=A	CLASS=IN	TTL=0	RDLENGTH=4
RDATA: 0.0.0.0					
Additional: TXT record (encrypted real answer)					
NAME=veildns.invalid.		TYPE=TXT	CLASS=IN	TTL=0	RDLENGTH
RDATA: context=random256 || tag || AEAD.Enc(ANCOUNT=1 || NSCOUNT=0 || ARCOUNT=0 || RR(example.com A IN TTL=300 192.0.2.1) || padding)					
Additional: OPT					
NAME=.		TYPE=OPT	CLASS=1232	TTL=0	RDLENGTH
RDATA: Code=10 || len=24 || ClientCookie || ServerCookie Code=65001 || len=1 || 0x01 (VeilDNS version)					

Fig. 3. **Stateless VeilDNS wire format.**

TABLE V
VeilDNS modes. Cached keys are master keys. † State retention is confirmed by the next authenticated query.

Mode	Warmup exchanges	State after warmup	KEM ops after warmup	Forward Secrecy
Stateless	0	None	Every query	No
Stateful	1	Cached keys†	None	No
Stateful-FS	2	Cached FS keys	None	From 2nd query

G. Threat model

We consider an active network adversary on the client-to-resolver path. The adversary can observe, drop, modify, inject, reorder, and replay DNS packets. It can also initiate VeilDNS sessions using the resolver's public KEM key.

The KEM public key is authenticated during setup. We trust the endpoints to protect local session state and execute the specified KEM and AEAD operations. The resolver is additionally trusted to resolve only decrypted DNS questions and erase ephemeral secrets in stateful-FS mode.

The privacy goal is that an adversary who has not compromised the resolver static KEM private key or directional master keys does not learn the real DNS question or any client-accepted answer. The authentication goal is that clients accept only resolver-generated responses bound to their queries; established stateful sessions also reject replayed queries and responses. For the stateful-FS mode, after the client accepts the resolver ephemeral key and the endpoints derive forward-secret state, later compromise of the resolver static KEM private key cannot reveal data protected in the second and later exchanges. This claim relies on prompt erasure of the ephemeral private key and intermediate shared secrets [27].

Endpoints reject VeilDNS packets if required records are missing or malformed, required KEM material cannot be processed, DNS Cookie validation fails, the referenced session state is unavailable or inconsistent, AEAD verification or protected question/answer parsing fails, AEAD-AD does not match the visible transcript, or a cached stateful session sees an invalid sequence number. § A-A gives the concrete prototype behavior for invalid packets, including diagnostic strings for authentication, replay, cookie, KEM, and parsing failures.

IV. Tamarin proof

Tamarin [30] is a tool for analyzing security protocols. It models each protocol step as a rule that updates protocol state and exchanges messages over a network controlled by a Dolev–Yao adversary [31]. The model treats the cryptographic primitives as secure and focuses on how the protocol combines them. As rules execute, they record security-relevant events called action facts. Each security property is stated as a lemma over these facts and checked across all possible executions.

The VeilDNS model captures the relevant elements: the authenticated resolver KEM key, transmitted `PRIVATE` and `TXT` records, KEM outputs, AEAD-protected payloads, session caches, directional master keys, one-time packet keys, stateful-FS transitions, and replay events. It abstracts byte-level DNS encoding, OPT, padding, parsing, and packet sizes. It also excludes randomness failures and side channels.

The stateless query and response rules use a fresh symbolic packet context for each encryption, modeling the random 256-bit values generated by the client and resolver. If an attacker replays a stateless query, the resolver derives the same response master key but a distinct one-time packet key. The stateful(-FS) exchange 1 rules generate a fresh symbolic sid_0 at the client and a fresh sid_1 at the resolver; both are included in the session master-key derivation. The model binds the visible DNS transaction ID and header flags in both AEAD directions. The response-authentication lemma matches these fields and the protected record sections across the resolver-send and client-accept action facts.

The dummy DNS question and answer are public, while the real question and answer appear only inside encrypted values. The model treats `QTYPE` and `QCLASS` as public constants and tracks `QNAME` separately. A receive rule succeeds only when the ciphertext matches the expected key and AEAD-AD, modeling tag verification and binding to the DNS exchange that carries it. Tamarin then proves that the real `QNAME` and client-accepted answers remain secret while the resolver static key and all directional master keys are uncompromised.

The lemma below directly proves `QNAME` secrecy unless the resolver static KEM key or any directional master key is revealed. An analogous lemma proves that the real answer in any client-accepted response remains secret under the same

Algorithm 1 VeilDNS online operations

Require: Resolver static KEM keypair and key identifier $(\mathsf{pk}_s, \mathsf{sk}_s, \mathsf{kid}_{pk_s})$.

Stateless

1: $\mathsf{C} : (\mathsf{ct}_s, \mathsf{ss}_s) \leftarrow \mathrm{Encap}(\mathsf{pk}_s)$; $(\mathsf{K}_{c2s}, \mathsf{K}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s; \mathsf{ct}_s, \mathsf{kid}_{pk_s})$; $\mathsf{k}_{c2s} \leftarrow \mathrm{KDF}(\mathsf{K}_{c2s})$.
2: $\mathsf{C} \rightarrow \mathsf{S} : \mathrm{Pkt}(\mathsf{private}{=}(\mathsf{kid}_{pk_s}, \mathsf{ct}_s), \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}_{c2s}}(\mathsf{Q}_{\mathrm{real}}))$.
3: $\mathsf{S} : \mathsf{ss}_s \leftarrow \mathrm{Decap}(\mathsf{sk}_s, \mathsf{ct}_s)$; $(\mathsf{K}_{c2s}, \mathsf{K}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s; \mathsf{ct}_s, \mathsf{kid}_{pk_s})$; $\mathsf{k}_{c2s} \leftarrow \mathrm{KDF}(\mathsf{K}_{c2s})$; $\mathsf{Q}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}_{c2s}}(\mathsf{txt})$.
4: $\mathsf{S} : \mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{S} \rightarrow \mathsf{C} : \mathrm{Pkt}(\mathsf{private}{=}\varnothing, \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}_{s2c}}(\mathsf{R}_{\mathrm{real}}))$.
5: $\mathsf{C} : \mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{R}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}_{s2c}}(\mathsf{txt})$.

Stateful cold-phase

6: C, S : run Stateless steps 1–3 with a fresh client-generated sid_0 included in the master-key KDF inputs.
7: $\mathsf{S} : \mathsf{sid}_1 \leftarrow \mathrm{random}(128)$; $(\mathsf{K}_{c2s}, \mathsf{K}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s; \mathsf{sid}_0, \mathsf{sid}_1, \mathsf{ct}_s, \mathsf{kid}_{pk_s})$; cache $(\mathsf{sid}_1; \mathsf{K}_{c2s}, \mathsf{K}_{s2c})$.
8: $\mathsf{S} : \mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{S} \rightarrow \mathsf{C} : \mathrm{Pkt}(\mathsf{private}{=}\varnothing, \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}_{s2c}}(\mathsf{R}_{\mathrm{real}}))$.
9: $\mathsf{C} : (\mathsf{K}_{c2s}, \mathsf{K}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s; \mathsf{sid}_0, \mathsf{sid}_1, \mathsf{ct}_s, \mathsf{kid}_{pk_s})$; $\mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{R}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}_{s2c}}(\mathsf{txt})$; cache $(\mathsf{sid}_1; \mathsf{K}_{c2s}, \mathsf{K}_{s2c})$.

Stateful warm-phase

10: $\mathsf{C} : \mathsf{k}_{c2s} \leftarrow \mathrm{KDF}(\mathsf{K}_{c2s})$; $\mathsf{C} \rightarrow \mathsf{S} : \mathrm{Pkt}(\mathsf{private}{=}\varnothing, \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}_{c2s}}(\mathsf{Q}_{\mathrm{real}}))$.
11: S : retrieve $(\mathsf{sid}_1; \mathsf{K}_{c2s}, \mathsf{K}_{s2c})$; $\mathsf{k}_{c2s} \leftarrow \mathrm{KDF}(\mathsf{K}_{c2s})$; $\mathsf{Q}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}_{c2s}}(\mathsf{txt})$.
12: $\mathsf{S} : \mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{S} \rightarrow \mathsf{C} : \mathrm{Pkt}(\mathsf{private}{=}\varnothing, \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}_{s2c}}(\mathsf{R}_{\mathrm{real}}))$.
13: $\mathsf{C} : \mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{R}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}_{s2c}}(\mathsf{txt})$.

Stateful-FS cold-phase

14: C, S : run Stateful steps 6–7, retaining the result as pending FS state.
15: $\mathsf{S} : (\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathrm{KEM}_e\,.\,\mathrm{KeyGen}()$; $\mathsf{kid}_{pk_e} \leftarrow \mathrm{KeyId}(\mathsf{pk}_e)$.
16: $\mathsf{S} : \mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{S} \rightarrow \mathsf{C} : \mathrm{Pkt}(\mathsf{private}{=}(\mathsf{kid}_{pk_e}, \mathsf{pk}_e), \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}_{s2c}}(\mathsf{R}_{\mathrm{real}}))$.
17: $\mathsf{C} : (\mathsf{K}_{c2s}, \mathsf{K}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s; \mathsf{sid}_0, \mathsf{sid}_1, \mathsf{ct}_s, \mathsf{kid}_{pk_s})$; $\mathsf{k}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}_{s2c})$; $\mathsf{R}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}_{s2c}}(\mathsf{txt})$.
18: $\mathsf{C} : (\mathsf{ct}_e, \mathsf{ss}_e) \leftarrow \mathrm{Encap}(\mathsf{pk}_e)$; $(\mathsf{K}^{\mathrm{FS}}_{c2s}, \mathsf{K}^{\mathrm{FS}}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s \| \mathsf{ss}_e; \mathsf{sid}_0, \mathsf{sid}_1, \mathsf{ct}_s, \mathsf{kid}_{pk_s}, \mathsf{pk}_e, \mathsf{ct}_e, \mathsf{kid}_{pk_e}, \texttt{"provisional"})$; $\mathsf{k}^{\mathrm{FS}}_{c2s} \leftarrow \mathrm{KDF}(\mathsf{K}^{\mathrm{FS}}_{c2s})$.
19: $\mathsf{C} \rightarrow \mathsf{S} : \mathrm{Pkt}(\mathsf{private}{=}(\mathsf{kid}_{pk_e}, \mathsf{ct}_e), \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}^{\mathrm{FS}}_{c2s}}(\mathsf{Q}_{\mathrm{real}}))$.
20: $\mathsf{S} : \mathsf{ss}_e \leftarrow \mathrm{Decap}(\mathsf{sk}_e, \mathsf{ct}_e)$; $(\mathsf{K}^{\mathrm{FS}}_{c2s}, \mathsf{K}^{\mathrm{FS}}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s \| \mathsf{ss}_e; \mathsf{sid}_0, \mathsf{sid}_1, \mathsf{ct}_s, \mathsf{kid}_{pk_s}, \mathsf{pk}_e, \mathsf{ct}_e, \mathsf{kid}_{pk_e}, \texttt{"provisional"})$.
21: $\mathsf{S} : \mathsf{k}^{\mathrm{FS}}_{c2s} \leftarrow \mathrm{KDF}(\mathsf{K}^{\mathrm{FS}}_{c2s})$; $\mathsf{Q}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}^{\mathrm{FS}}_{c2s}}(\mathsf{txt})$; $\mathsf{k}^{\mathrm{FS}}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}^{\mathrm{FS}}_{s2c})$.
22: $\mathsf{S} \rightarrow \mathsf{C} : \mathrm{Pkt}(\mathsf{private}{=}\varnothing, \mathsf{txt}{=}\,\mathrm{Encrypt}_{\mathsf{k}^{\mathrm{FS}}_{s2c}}(\mathsf{R}_{\mathrm{real}}))$.
23: $\mathsf{C} : \mathsf{k}^{\mathrm{FS}}_{s2c} \leftarrow \mathrm{KDF}(\mathsf{K}^{\mathrm{FS}}_{s2c})$; $\mathsf{R}_{\mathrm{real}} \leftarrow \mathrm{Decrypt}_{\mathsf{k}^{\mathrm{FS}}_{s2c}}(\mathsf{txt})$.
24: $\mathsf{C}, \mathsf{S} : (\mathsf{K}^{\mathrm{FS}}_{c2s}, \mathsf{K}^{\mathrm{FS}}_{s2c}) \leftarrow \mathrm{KDF}(\mathsf{ss}_s \| \mathsf{ss}_e; \mathsf{sid}_0, \mathsf{sid}_1, \mathsf{ct}_s, \mathsf{kid}_{pk_s}, \mathsf{pk}_e, \mathsf{ct}_e, \mathsf{kid}_{pk_e}, \texttt{"final"})$; cache $(\mathsf{sid}_1; \mathsf{K}^{\mathrm{FS}}_{c2s}, \mathsf{K}^{\mathrm{FS}}_{s2c})$.

Stateful-FS warm-phase

25: C, S : run Stateful steps 10–13 with $(\mathsf{K}_{c2s}, \mathsf{K}_{s2c}) = (\mathsf{K}^{\mathrm{FS}}_{c2s}, \mathsf{K}^{\mathrm{FS}}_{s2c})$.

Notation. C/S are client/resolver; sid_0 and sid_1 denote the initial and established session identifiers.
$\mathrm{KDF}(\mathsf{ss}; \ldots)$ derives master keys $(\mathsf{K}_{c2s}, \mathsf{K}_{s2c})$; `"provisional"`/`"final"` denote key phases; FS denotes forward secrecy.
For each packet, $\mathsf{k} \leftarrow \mathrm{KDF}(\mathsf{K})$ denotes derivation of a one-time AEAD key and nonce from master key K and the packet context; only the key is shown.
$\mathrm{Pkt}(\mathsf{private}, \mathsf{txt})$ denotes the dummy DNS packet with an optional `PRIVATE` record and encrypted `TXT` record; DNS Cookie processing is omitted.
$\mathsf{Q}_{\mathrm{real}}$ is the padded question; $\mathsf{R}_{\mathrm{real}}$ is the padded sequence of response records.

conditions. In the listings, `All` and `Ex` denote universal and existential quantification, `#i` identifies an event time, and `K(x)` means that the adversary knows x.

```
lemma question_secrecy:
  "All qn #i.
     /* If qn is a protected QNAME, */
     SecretQuery(qn) @ #i
     /* and the resolver static key is unrevealed, */
   & not (Ex sk #j. ServerSecretReveal(sk) @ #j)
     /* and no directional master key is revealed, */
   & not (Ex mode sid dir k #j.
            SessionKeyReveal(mode, sid, dir, k) @ #j)
     /* then the adversary never learns the QNAME. */
   ==> not (Ex #k. K(qn) @ #k)"
```

The excerpt below shows how a receive rule constructs the packet context and one-time key, then accepts `qplain` only when the ciphertext matches the expected AEAD-AD over the mode, direction, context, transaction ID, and query flags:

```
ctx = packetctx(sid, 'c2s', seq)
packet_k = packetkey(kc2s, mode, 'c2s', ctx)
txt = aead(packet_k,aad(mode,'c2s',ctx,txid,qflags),qplain)
```

A replay-tracked receive emits an action fact recording the accepted sequence number and key phase:

```
--[ ServerReplayCache(mode, sid, phase, 'c2s', seq) ]->
```

The replay-cache restriction permits at most one acceptance for each mode, session, key phase, direction, and sequence:

```
restriction StatefulServerReplayCache:
  "All mode sid phase dir seq #i #j.
     /* If a sequence is accepted twice under the same
        mode, session, key phase, and direction, */
     ServerReplayCache(mode, sid, phase, dir, seq) @ #i
   & ServerReplayCache(mode, sid, phase, dir, seq) @ #j
```

TABLE VI
TAMARIN PROOF PROPERTIES AND SCOPE.

Property	Coverage	Result and scope
Query secrecy	1 lemma; all modes	The real QNAME remains secret even when QTYPE and QCLASS are public, unless the static KEM private key or any directional master key is revealed.
Accepted-answer secrecy	1 lemma; all modes	The real answer in any accepted response remains secret unless the static KEM private key or any directional master key is revealed.
Response authentication	1 lemma; all modes	Accepted response flags and protected answer records match those sent by the resolver unless the static KEM private key or a master key is revealed.
Resolver-to-client key secrecy	1 lemma; all modes	The resolver-to-client master key remains secret unless that key or the static KEM private key is revealed.
Packet-key non-reuse	1 lemma; all modes	No one-time packet key appears in two honest encryption events under the modeled freshness and replay assumptions.
Replay rejection	2 restrictions, 4 lemmas	Authenticated sequences and replay-cache uniqueness prevent replay-tracked stateful(-FS) queries and responses from being accepted twice.
Transcript binding	Built into receive rules	AEAD binds each message to its mode, direction, packet context, DNS transaction ID, and header flags.
Stateful-FS ephemeral auth.	1 lemma; stateful-FS	Clients accept only resolver-generated ephemeral public keys unless the static KEM private key or a master key is revealed.
Stateful-FS forward secrecy	2 lemmas; stateful-FS	From exchange 2, FS master keys, QNAMEs, and client-accepted answers remain secret after later static-key compromise under the stated assumptions.

```
  /* then both facts must be the same event. */
==> #i = #j"
```

Since the seq is AEAD-authenticated, an attacker cannot alter it without causing AEAD verification to fail; the uniqueness restriction rejects an exact replay under its original seq. Under these replay-cache restrictions, four lemmas establish client- and server-side replay rejection for replay-tracked stateful traffic. These lemmas assume the restrictions rather than model the concrete replay-cache state machine.

Stateful-FS is modeled in three phases: the static-KEM first exchange, the ephemeral-KEM second exchange, and the final cached state. This lets the proof distinguish the non-forward-secret first query from the forward-secret second and later queries. The transition to the final state consumes the resolver's pending state and assumes that the ephemeral private key and intermediate shared secrets are unavailable afterward. Table VI summarizes the proved properties and their scope.

On an M4 MacBook Pro with 16 GB RAM, Tamarin 1.12.0 verified all 12 security lemmas in approximately 8.2 minutes.

V. EVALUATION

We evaluate VeilDNS against vanilla DNS and post-quantum DoX baselines with two broad experiments. Cold-phase benchmarks test DNS lookups that begin with no cached VeilDNS state and no existing DoX channel. Warm-phase benchmarks test the post-warmup case, where stateful sessions and persistent DoX connections reuse established state across DNS queries. Vanilla DNS and the VeilDNS stateless mode do not create session state.

A. Implementation

The prototype runs as a Docker testbed that places all experimental behavior inside two proxy containers. As shown in Figure 4, the client container first sends a standard DNS/UDP query to the client proxy. The client proxy and server proxy

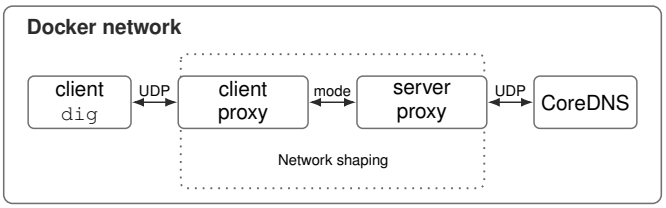


Fig. 4. **Docker testbed.** VeilDNS, DoX, or vanilla DNS runs between the proxies; the dotted box marks where network profiles are applied.

then carry the selected VeilDNS, DoX, or vanilla DNS mode over the proxy-to-proxy hop. Finally, the server proxy forwards an ordinary DNS/UDP query to a CoreDNS backend. In VeilDNS modes, the proxies handle DNS Cookies while preserving other OPT fields and options during forwarding.

Network shaping is applied only to the proxy-to-proxy hop. We use two profiles: a high-bandwidth profile with 100 Mbps bandwidth and 10 ms one-way delay, and a low-bandwidth profile with 1 Mbps bandwidth and 100 ms one-way delay. The profiles are implemented with Linux tc inside the proxy containers, and are scoped to traffic between the two proxies. The client-to-client-proxy and server-proxy-to-CoreDNS links are left unshaped, with no added delay or bandwidth cap.

The client records latency with wall-clock timers around DNS send/receive. Byte and packet counts are proxy-to-proxy IP-layer measurements. Warm-phase runs read the IPv4 total_length field from raw-socket packet logs inside the proxy containers. Cold-phase runs use tc -s qdisc egress byte and packet counter deltas. Proxy CPU time is measured with perf stat task-clock.

CoreDNS 1.14.3 serves a local zone; benchmarks cycle through ten single-answer A queries and count only expected answers as successful. All proxy paths are implemented in C; vanilla forwards DNS unchanged. The DoX baselines use OpenSSL's TLS and QUIC APIs, with local code for DNS

TABLE VII
ALGORITHM COMPARISON. SIZE COLUMNS REPORT RAW PUBLIC KEYS, CIPHERTEXTS, OR SIGNATURES. OPERATION RATES ARE MEDIANS OVER THREE RUNS OF THREE SECONDS EACH.

Algorithm	PQ?	pk	ct/sig	Op/s
X25519	No	32	32	55k derive/s
ML-KEM-512	Yes	800	768	141k encap/s, 121k decap/s
HQC-1	Yes	2241	4433	792 encap/s, 529 decap/s
ECDSA P-256	No	64	64	89k sign/s, 30k verify/s
RSA-2048	No	256	256	3k sign/s, 116k verify/s
Falcon-512	Yes	897	752	7k sign/s, 53k verify/s
ML-DSA-44	Yes	1312	2420	16k sign/s, 43k verify/s
SLH-DSA-SHA2-128s	Yes	32	7856	4 sign/s, 4k verify/s

framing, HTTP/2 DoH handling, and DoQ stream handling. Warm DoT reuses one pipelined connection, while warm DoH and DoQ use one stream per query over a reused connection. PQC support is provided by OQS [32]. The cryptographic stack is OpenSSL 3.5.7, `liboqs` 0.16.0, and `oqs-provider` 0.12.0-dev at commit `b104f576`. The local experiments are run on an M4 MacBook Pro (16 GB RAM).

B. Algorithm Choice

The main evaluation targets NIST security category 1 using stable, standardized primitives. All evaluated protocol configurations use AES-256-GCM [33] and HKDF [34].

VeilDNS uses ML-KEM-512 [35] as its KEM and zero-pads plaintext before encryption, first to at least 128 bytes and then in 64-byte buckets. HQC artifacts exceed the DNS/UDP size budget and would cause IP fragmentation on typical network links, so HQC is excluded from the main evaluation.

The DoX baselines also use ML-KEM-512 for key exchange. The primary DoX baselines use ML-DSA-44 [36] certificates; the secondary DoX baselines use Falcon-512 [37] and SLH-DSA-SHA2-128s [38]. All use a self-signed certificate without a certificate chain, so their reported handshake costs represent a best-case scenario for DoX.

Table VII summarizes raw artifact sizes and local Docker throughput. Classical algorithms use OpenSSL APIs, and post-quantum algorithms use liboqs APIs.

C. Cold-phase performance

A cold session starts with no prior VeilDNS state and no existing DoX channel or usable TLS or QUIC resumption state. The stateful and stateless cold paths have similar cryptographic processing and wire shape: each carries a fresh ML-KEM ciphertext and encrypted DNS question, while stateful additionally caches the derived master keys. We therefore report one stateful/less row, measured using stateful mode. Stateful-FS adds a second UDP exchange with an already forward-secret DNS question/answer, after which both endpoints cache the final FS master keys. Each DoX baseline establishes a fresh TLS/TCP or QUIC channel before returning the DNS answer. Vanilla DNS is the plain DNS/UDP reference case. Thus, a cold session involves one UDP exchange for vanilla DNS and stateful/less, two for stateful-FS, and a TLS/TCP or QUIC channel setup plus a DNS exchange for DoX.

We report capacity-point throughput, fixed-load latency, IP-layer bytes and packets (client-proxy view), and proxy CPU time. Throughput is the completion rate of the configured finite session batch at its selected capacity preset; latency reports client-observed average and 95th-percentile DNS resolution time at common fixed load. Byte and packet counts expose the proxy-to-proxy network cost of each cold path, while proxy CPU shows the local computational work required to complete the cold-session processing for each mode.

All modes are measured after setup. In our VeilDNS prototype, the resolver ML-KEM public key is pinned in the proxy configuration. For DoX, a self-signed resolver certificate is installed as trusted testbed material. These setup costs remain outside the measurements.

Experiment. We run 500 fresh sessions per mode and network profile. Concurrency c denotes the number of active cold sessions. Throughput is measured at preset 100%-success capacity points c_{CP} selected from prior capacity sweeps,[9] while latency, byte counts, packet counts, and CPU are measured separately at common fixed loads c_{FL}. Byte and packet counts are per-session averages from aggregate proxy-to-proxy IP-layer counters, so packet counts can be fractional. For presentation, byte counts are rounded to the nearest integer.

Results. Figure 5 shows the high-bandwidth results, and Table VIII gives the full high- and low-bandwidth profile numbers. The table below compares VeilDNS and DoX.

		DoT					DoH					DoQ				
Profile	VeilDNS	T	L	B	P	C	T	L	B	P	C	T	L	B	P	C
High	Stateful/less	3.1	3.2	7.9	13.6	4.1	3.2	3.1	8.3	15.8	4.1	3.5	4.8	13.7	13.5	4.7
High	Stateful-FS	1.5	1.7	3.0	6.8	2.2	1.6	1.7	3.2	7.9	2.2	1.7	2.6	5.3	6.7	2.5
Low	Stateful/less	9.1	4.7	8.1	15.2	6.6	9.3	4.8	8.6	18.9	6.8	12.4	5.9	14.0	13.8	7.5
Low	Stateful-FS	4.5	2.3	3.1	7.6	3.3	4.7	2.4	3.3	9.4	3.4	6.2	2.9	5.4	6.9	3.8

VeilDNS vs DoX comparison. Ratios are computed from Table VIII: VeilDNS **T**hroughput is T times higher, while fixed-load average **L**atency, total **B**yte count, total **P**acket count, and total proxy **C**PU time are L, B, P, and C times lower than each DoX.

Remarks. VeilDNS beats DoX on every reported metric, with the largest gains under the low-bandwidth profile where DoX's handshake-heavy traffic shape has the greatest impact. On the high-bandwidth profile, the stateful/less modes are practically equivalent to vanilla DNS in throughput and latency. On the low-bandwidth profile, vanilla DNS retains higher throughput because its packets are much slimmer. The stateful-FS mode achieves about half the cold throughput of the stateful/less VeilDNS modes, while still exceeding the fastest DoX baseline by $1.5\times$ in High and $4.5\times$ in Low.

DoQ incurs higher byte and latency costs than DoT and DoH because QUIC pads Initial datagrams to at least 1200 bytes and limits pre-validation server traffic to roughly $3\times$ the bytes the server has received from the client. In our testbed this yields 1228-byte IPv4 packets and only about 3.6 KB of initial server send budget; the post-quantum TLS

[9] A capacity sweep varies c for a fixed-size session batch and selects the highest-throughput concurrency at which all sessions complete successfully.

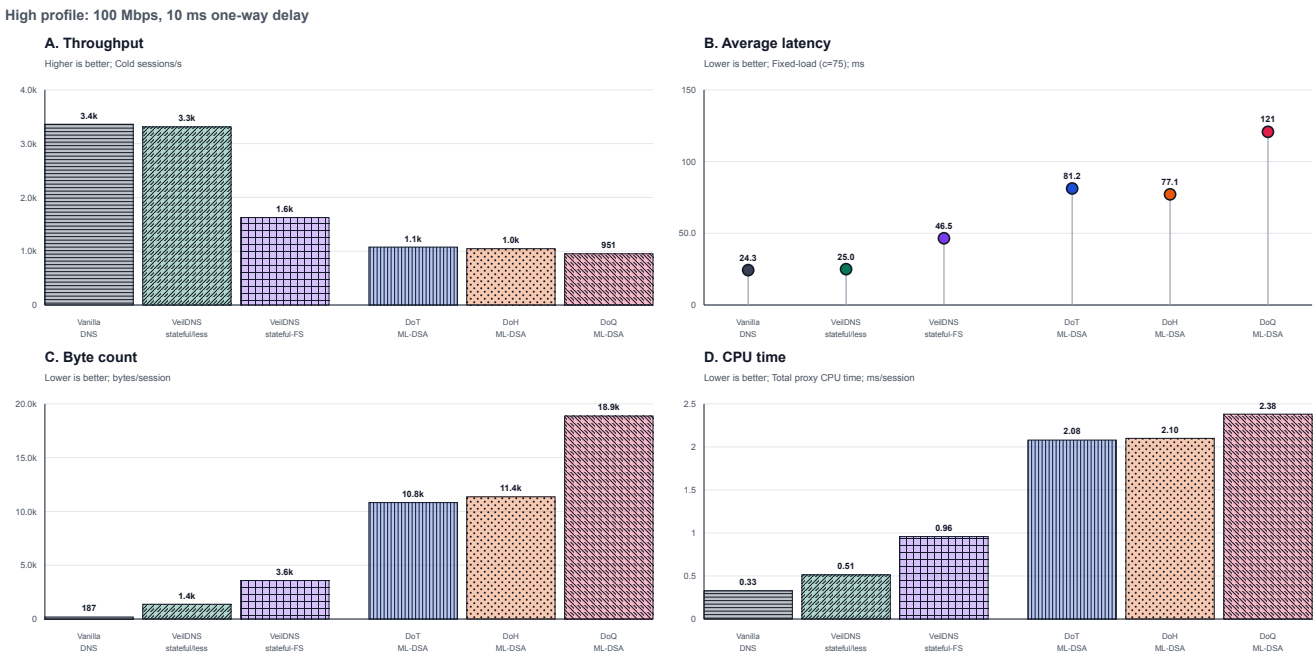


Fig. 5. **Cold-phase benchmark, high-bandwidth profile.** Plotted from the High-profile rows in Table VIII.

TABLE VIII
COLD-PHASE BENCHMARK. BANDWIDTH PROFILES: HIGH = 100 MBPS, 10 MS ONE-WAY DELAY; LOW = 1 MBPS, 100 MS ONE-WAY DELAY. THROUGHPUT USES CAPACITY-POINT CONCURRENCY c_{CP}; LATENCY, BYTES, PACKETS, AND CPU USE FIXED-LOAD CONCURRENCY c_{FL}.

		Throughput			Latency (ms)		Bytes/session			Packets/session			CPU ms/session		
Profile	Mode	c_{CP}	Sessions/s	c_{FL}	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
High	Vanilla DNS	200	3360.2	75	24.3	46.4	80	107	187	1.0	1.0	2.0	0.29	0.04	0.33
High	VeilDNS stateful/less	175	3309.8	75	25.0	54.8	1063	310	1373	1.0	1.0	2.0	0.39	0.13	0.51
High	VeilDNS stateful-FS	175	1625.6	75	46.5	72.4	2142	1437	3579	2.0	2.0	4.0	0.70	0.26	0.96
High	DoT ML-DSA	150	1076.4	75	81.2	121.8	2041	8781	10821	14.0	13.1	27.1	0.96	1.12	2.08
High	DoH ML-DSA	150	1046.3	75	77.1	105.2	2347	9012	11359	16.3	15.3	31.6	0.98	1.12	2.10
High	DoQ ML-DSA	200	951.1	75	120.7	131.6	7994	10878	18871	13.9	13.0	26.9	1.29	1.09	2.38
Low	Vanilla DNS	200	699.2	10	212.1	216.6	80	107	187	1.0	1.0	2.0	0.75	0.12	0.88
Low	VeilDNS stateful/less	25	110.0	10	221.8	226.5	1063	310	1373	1.0	1.0	2.0	0.98	0.46	1.44
Low	VeilDNS stateful-FS	25	54.9	10	449.3	456.8	2142	1437	3579	2.0	2.0	4.0	1.95	0.91	2.86
Low	DoT ML-DSA	25	12.1	10	1052.3	1064.0	2144	8942	11086	15.7	14.7	30.4	4.16	5.34	9.50
Low	DoH ML-DSA	25	11.8	10	1064.9	1066.0	2570	9244	11814	19.9	17.9	37.7	4.38	5.47	9.86
Low	DoQ ML-DSA	25	8.9	10	1313.3	1316.2	8131	11033	19164	14.2	13.4	27.6	4.82	6.01	10.83

handshake exceeds that budget, forcing extra padded handshake round trips [15].

Certificate sensitivity. Table IX isolates the effect of the DoX certificate on high-bandwidth cold sessions. These runs keep the same ML-KEM-512 key exchange, but replace the primary ML-DSA-44 certificate algorithm with Falcon-512 or SLH-DSA-SHA2-128s. Falcon reduces transport bytes and improves cold throughput relative to ML-DSA, while SLH-DSA collapses throughput to just tens of sessions/s because signing dominates server-proxy CPU. Thus, Falcon mainly improves DoX throughput and network cost; SLH-DSA materially worsens performance; and VeilDNS still achieves higher cold-session throughput than all measured DoX certificate variants because it avoids transport channel establishment.

D. Warm-phase performance

Warm-phase benchmarks measure recurring-query performance after mode-specific warmup. The stateful and stateful-FS modes require one and two UDP warmup exchanges, respectively, to install cached master keys; their AEAD processing and wire shape are identical, so we report one stateful(-FS) row. The stateless mode creates no session state; measured queries echo a previously obtained Server Cookie and carry a fresh ML-KEM ciphertext. DoX baselines establish their

TABLE IX
COLD-PHASE CERTIFICATE SENSITIVITY IN HIGH-BANDWIDTH PROFILE. THROUGHPUT USES CAPACITY-POINT CONCURRENCY c_{CP}; LATENCY, BYTES, PACKETS, AND CPU USE FIXED-LOAD CONCURRENCY c_{FL}.

		Throughput			Latency (ms)		Bytes/session			Packets/session			CPU ms/session		
Transport	Signature	c_{CP}	Sessions/s	c_{FL}	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
DoT	Falcon	150	1198.3	10	65.6	69.5	1744	4615	6358	12.0	10.0	22.0	1.29	1.78	3.07
DoT	ML-DSA	150	1000.7	10	65.3	69.4	2039	8777	10817	14.0	13.0	27.0	1.13	1.46	2.59
DoT	SLH-DSA	20	22.4	10	480.7	512.5	2053	18737	20791	18.0	21.0	39.0	2.97	411.27	414.24
DoH	Falcon	150	1135.1	10	65.5	71.9	2037	4834	6871	14.1	12.0	26.1	1.34	1.80	3.14
DoH	ML-DSA	150	972.7	10	64.5	66.1	2332	8997	11329	16.0	15.0	31.1	1.14	1.53	2.67
DoH	SLH-DSA	20	22.4	10	480.2	513.1	2346	18963	21309	20.0	23.2	43.2	3.03	410.80	413.83
DoQ	Falcon	200	1247.1	10	71.8	77.8	3954	4034	7988	6.1	6.0	12.1	1.72	2.02	3.74
DoQ	ML-DSA	200	939.2	10	91.5	97.5	7892	10137	18029	12.5	12.4	25.0	1.75	2.07	3.83
DoQ	SLH-DSA	20	21.0	10	713.0	1331.2	4358	18423	22781	11.5	18.0	29.5	2.98	310.51	313.49

TABLE X
WARM-PHASE BENCHMARK. CONCURRENCY $c = 5$; QPS DENOTES QUERIES PER SECOND.

		Throughput	Latency (ms)		Bytes/query			Packets/query			CPU ms/query		
Profile	Mode	QPS	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
High	Vanilla DNS	200.7	24.8	28.4	80	107	187	1.0	1.0	2.0	1.11	0.12	1.24
High	VeilDNS stateless	199.3	25.0	27.3	1079	310	1389	1.0	1.0	2.0	1.12	0.28	1.40
High	VeilDNS stateful(-FS)	204.5	24.4	26.6	294	310	604	1.0	1.0	2.0	0.95	0.19	1.14
High	DoT ML-DSA	215.6	23.0	25.0	138	161	299	1.4	1.3	2.8	0.75	0.63	1.39
High	DoH ML-DSA	215.1	22.9	24.5	206	208	414	1.4	1.4	2.8	0.76	0.69	1.45
High	DoQ ML-DSA	216.3	23.0	24.9	132	150	282	1.5	1.5	3.0	0.79	0.69	1.48
Low	Vanilla DNS	23.4	213.7	218.1	80	107	187	1.0	1.0	2.0	1.60	0.26	1.86
Low	VeilDNS stateless	22.5	221.6	226.9	1079	310	1389	1.0	1.0	2.0	1.50	0.65	2.16
Low	VeilDNS stateful(-FS)	23.1	216.1	221.2	294	310	604	1.0	1.0	2.0	1.41	0.48	1.89
Low	DoT ML-DSA	23.6	211.6	215.1	135	160	295	1.4	1.3	2.7	1.49	1.42	2.92
Low	DoH ML-DSA	23.6	211.9	214.3	202	201	403	1.3	1.3	2.6	1.37	1.23	2.60
Low	DoQ ML-DSA	23.6	211.6	214.4	139	154	293	1.6	1.6	3.1	1.91	1.55	3.47

transport connection before measurement, while vanilla DNS, like stateless mode, has no warmup.

We report the number of queries resolved per second (QPS), average and p95 latency, IP-layer bytes and packets (client-proxy view), and proxy CPU time per successful query. QPS and latency show sustained service at fixed concurrency. Byte and packet counts expose the recurring traffic shape, while CPU time shows the per-query proxy compute work.

Experiment. For each mode and bandwidth-profile pair, we run a batch of 100 DNS queries at concurrency $c = 5$; here, c denotes the number of outstanding DNS queries. Byte and packet counts are per-query averages from aggregate proxy-to-proxy IP-layer counters, so packet counts can be fractional. For presentation, byte counts are rounded to the nearest integer.

Results. Table X reports the warm-phase numbers.

Remarks. Once warm, VeilDNS and DoX cluster near vanilla DNS in throughput and latency. This is the expected contrast with the cold-phase result: after the stateful(-FS) modes cache their keys and DoX baselines reuse their channels, the cold-start cost is no longer on the query path. In byte count, the stateless mode remains an exception because it carries a fresh KEM ciphertext on every query.

The stateful(-FS) modes remove the recurring KEM ciphertext, reducing per-query traffic relative to stateless mode. Compared with vanilla DNS and the warm DoX rows, the stateful(-FS) modes have some residual byte overhead because messages still carry a dummy DNS question/answer, a `TXT` record containing a packet context, AEAD tag, and padded ciphertext, and DNS Cookies in an `OPT` record.

Certificate sensitivity. Changing the DoX certificate algorithm does not affect warm DoX performance. After the handshake, the warm-path uses only symmetric operations.

E. Additional Measurements

Lossy link. We repeated the local experiments with 5% packet loss on the proxy-to-proxy path, using 500 cold sessions and three 100-query warm batches per mode. § B-A gives the detailed setup, and Tables XIII and XIV report the results. With proxy retries enabled, all vanilla DNS and VeilDNS UDP operations completed; in warm operation, TCP head-of-line blocking increased DoT and DoH delay, while DoQ's independent streams avoided it.

Cloud. We repeated the cold- and warm-phase experiments over a SFO3–NYC1 cloud path with roughly 67 ms RTT, using

500 cold sessions and three 100-query warm batches per mode. § B-B gives the detailed setup, and Tables XV and XVI report the results. For throughput and latency, the cloud experiments reproduce the local trend: the stateful/less modes remain close to vanilla DNS for cold sessions and avoid the larger cold-start costs of DoX, while all VeilDNS modes remain comparable to warmed DoX for recurring queries.

F. Takeaway

The main benefit of VeilDNS is cold-phase performance. When resolver infrastructure has no prior per-client state, the protocol lets the same infrastructure support practically as many fresh private DNS lookups as vanilla DNS on well-provisioned networks. This is the deployment case where DoX is most expensive: each new client must first establish an authenticated transport channel before it can receive a DNS answer. As the network becomes constrained, that extra channel-establishment work dominates, and the throughput gap between VeilDNS and DoX becomes much larger.

The certificate-sensitivity results reinforce that VeilDNS's cold-start advantage comes from its design: changing the DoX signature algorithm can change the absolute handshake cost, but VeilDNS avoids transport-channel establishment altogether. Its cold path is a DNS exchange carrying AEAD-encrypted data and KEM artifacts, not a full TLS/TCP or QUIC handshake followed by DNS query/response. Even exchanges that are intended for state warmup are not empty setup trips: they carry encrypted DNS questions and answers while installing symmetric keys at endpoints.

This also makes reconnection cost central to the comparison. A DoX connection can break for ordinary operational reasons: idle timeout, mobile sleep, NAT/firewall state expiry, or network change. QUIC connection migration can preserve a channel across some path changes when endpoint and path state permit [15], [16]. When no usable resumption state remains, the next DNS lookup follows the cold path and rebuilds the TLS/TCP or QUIC channel. TLS or QUIC resumption and DoQ 0-RTT can reduce this cost when valid ticket state is retained, with associated replay and linkability considerations [11], [39]. In contrast, VeilDNS has no transport connection to maintain. The stateless mode requires no state by design, and the stateful(-FS) modes keep their cached master keys independent of any live connection; as long as cached state remains available and the DNS Cookies remain valid, later DNS/UDP transactions can continue without paying a transport reconnection cost.

The warm-phase results show that VeilDNS keeps this cold-start advantage without giving up recurring-query performance. Once warmup is complete, it remains close to vanilla DNS and DoX in throughput, latency, packet count, and CPU time while preserving the same UDP query/response shape. The stateful(-FS) modes retain some residual byte overhead from VeilDNS framing. The stateless mode has a higher recurring byte cost because every query carries a fresh KEM ciphertext to avoid resolver-side state.

VI. Discussion

A. Use Cases

The stateless mode is the lowest-friction deployment option since it requires no synchronized resolver state. It fits anycast resolvers because any instance with the active KEM private key can answer the next query. The stateful mode is the performance-oriented default. When warm, it uses less bandwidth and computation than stateless mode, making it the natural fit for unicast resolvers. Anycast deployments can keep state local with client-affinity routing or synchronize cached state across resolver instances. The stateful-FS mode fits high-risk use cases where DNS lookups are unusually sensitive, such as whistleblowing, journalism, or political dissidence.

B. Resolver-to-ANS Hop

This paper applies VeilDNS to the client-to-resolver hop, but the same approach can work on the resolver-to-authoritative name server (ANS) hop for resolver query privacy and ANS authentication. Specifically, DNSSEC can authenticate the zone KEM key used by the ANS: the zone can publish the KEM key in a `DNSKEY` record or bind it through a `TLSA` record [18]. A natural direction is to use VeilDNS with SL-DNSSEC [19], where the same zone KEM key can be used for both encryption and signature-less MAC-based DNSSEC validation of returned DNS records.

C. Resolver Key Rotation

Resolver KEM keys can rotate with certificate validity periods, `DNSKEY` or `TLSA` TTLs, or preconfiguration. During rotation, a resolver generates another key if its identifier collides with that of a previous key that clients may still have cached. If a resolver retires a KEM key before clients refresh their key view, stateless queries using that stale key receive a `BAD_KEM_ID` diagnostic response (§ A-A); recovery requires rerunning the setup phase. Warm stateful(-FS) sessions can continue under cached symmetric keys after rotation until their state expires under deployment policy or is lost. After static-key compromise, stateful sessions derived from that key must be discarded; established stateful-FS sessions remain protected if ephemeral and intermediate secrets were erased.

D. Protocol Discovery

VeilDNS support can be advertised through the same trust channels used to authenticate the resolver key: TLS certificate extensions, DNSSEC-validated metadata, and resolver profiles. A deployment can also reuse the SVCB-based resolver-discovery pattern standardized by Discovery of Designated Resolvers, extended with metadata for supported modes, algorithm suites, and resolver KEM key identifiers [40].

E. UDP Size Constraints

VeilDNS inherits DNS/UDP size constraints. Although DNS messages can be much larger in theory, VeilDNS uses a conservative 1232-byte DNS-message bound, matching common resolver defaults and IPv6-minimum-MTU arithmetic [5]. VeilDNS queries stay below the corresponding 1260-byte IPv4

packet size limit[10], but large responses may exceed the path MTU and risk fragmentation. For such responses, deployments could use the usual DNS/TCP fallback by returning the truncation `TC` bit with an AEAD tag over empty plaintext; the client retries the VeilDNS query over TCP only after verifying the tag, which authenticates `TC` through the AEAD-AD. Alternatively, deployments could switch to DoX.

The UDP size budget also constrains KEM choice. ML-KEM-768 would exceed the conservative 1232-byte DNS-message bound but remain within a typical 1500-byte path MTU, while ML-KEM-1024 would require a larger MTU. Future KEMs remain usable when their public keys and ciphertexts fit within the path's DNS/UDP budget.

F. DDoS Considerations

DNS amplification. VeilDNS uses an echoed Server Cookie to validate prior client reachability at the source address before accepting later queries [8]. This mitigates off-path DNS amplification attacks in which an attacker spoofs a victim's source address to direct large DNS responses toward it. During first contact, when a Server Cookie may not yet be available, deployments should use response rate/size limiting; ingress filtering [41] can also reduce spoofing. Moreover, the stateless mode has a much lower response-to-query size ratio because each query carries a KEM ciphertext, reducing the bandwidth amplification available from a spoofed query.

Server resource exhaustion. Cold VeilDNS exchanges require the resolver to decapsulate a KEM ciphertext, so spoofed first-contact queries can consume resolver CPU and create pending state before DNS Cookie validation. The prototype bounds and expires pending state before treating a session as established (§ A-C3). Warm stateful modes use cached symmetric keys without KEM decapsulation; deployments can additionally use rate limiting to bound first-contact CPU work.

G. Limitations

We evaluate VeilDNS in a controlled local Docker testbed and validate the observed trends on a cloud path. The protocol inherits UDP concerns such as path MTU variation and middlebox behavior. Its privacy scope is client-to-resolver DNS question and answer data. Packet timing, direction, source address, coarse packet sizes, outer DNS flags, EDNS metadata, stateful session identifiers, and sequence numbers remain visible. Padding reduces direct question/answer length leakage, but encrypted DNS traffic can still leak through size and timing correlations [42], [43], [44]. Resolver logging and data-sharing policies remain separate operator issues [45]. Evaluating large-scale resolver deployment remains outside this paper's scope.

VII. Related Work

Prior work studies the deployment, privacy, and performance of DoT, DoH, and DoQ [12], [13], [14], [46], [47]. DNS-over-DTLS [48] is an experimental encrypted DNS transport over UDP/853. It still requires DTLS session establishment, inherits datagram loss and fragmentation concerns, and has seen little practical deployment [11]. Accordingly, our transport comparison focuses on standards-track DoX.

T-DNS, a precursor to standardized DoT, proposed persistent DNS over TCP with TLS, pipelining, out-of-order responses, and connection reuse to amortize setup costs [49]. Oblivious DNS and Oblivious DoH separate client identity from query contents through a relay and encrypted target queries, assuming the relay and target do not collude [50], [51]. VeilDNS protects a direct client-to-resolver exchange and does not hide the client IP address from the resolver. QNAME minimization reduces how much of the queried domain name is revealed to authoritative servers, but the recursive resolver still sees the full name [52].

HPKE [53] standardizes a KEM–KDF–AEAD construction with labeled domain separation and nonce-managed encryption state; VeilDNS is not an HPKE instantiation and defines its own DNS-specific bidirectional keying, replay handling, DNS Cookie integration, and forward secrecy. KEMTLS [17] replaces TLS handshake signatures with KEM-based authentication, but still establishes a TLS channel. PQConnect is a post-quantum UDP tunnel with DNS-based discovery, long-term and ephemeral KEMs, and forward secrecy [54]. It protects subsequent host-to-host IP traffic; VeilDNS instead protects client-to-resolver DNS from the first UDP query.

Two expired Internet-Drafts previously explored encrypted DNS/UDP: Confidential DNS [55] used an unassigned `ENCRYPT` record in standard DNS framing, while Zuo *et al.* [56] used nonstandard flag-and-length framing. Neither provides VeilDNS's authenticated post-quantum design, three operating modes, online FS transition, or formal analysis.

DNSCrypt [57] avoids TLS and runs over UDP, but uses a custom encrypted packet format over UDP/443 and relies on its own provider-based PKI, while VeilDNS uses standard DNS packet format over UDP/53 and can authenticate the resolver KEM key through any PKI. DNSCrypt v2 is classical; a July 2026 Internet-Draft defines PQDNSCrypt by placing an X-Wing key in `resolver-pk` and a fresh X-Wing ciphertext in `client-pk`, while retaining Ed25519 provider authentication [58]. PQDNSCrypt supports stateless resumption tickets. Forward secrecy follows resolver-key rotation for ciphertext-bearing queries and ticket-key destruction for resumed traffic, whereas VeilDNS stateful-FS uses an online ephemeral KEM.

DNSCurve [59] provides DNS/UDP encryption for the resolver-to-ANS hop. It discovers keys through `NS` records, with Curve25519 keys encoded in name-server names, and uses a DNSCurve-specific wire format. A later post-quantum DNSCurve sketch replaces ECDH with a KEM and uses a server-encrypted cookie to split the oversized exchange [60]. EncDNS [61] builds on this DNSCurve style but follows an oblivious DNS model in which recursive resolvers relay encrypted client queries to an EncDNS service that performs resolution. VeilDNS instead protects the direct client-to-resolver hop and provides three modes, including a forward-secret mode using an online ephemeral KEM.

[10] 1232 B (DNS message) + 8 B (UDP header) + 20 B (IPv4 header).

AVAILABILITY

We intend to provide the Tamarin model, implementation, and evaluation scripts in a future revision of this paper.

REFERENCES

[1] P. E. Hoffman and K. Fujiwara, "DNS Terminology," RFC 9499, 2024.
[2] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels, "DNS Transport over TCP - Implementation Requirements," RFC 7766, 2016.
[3] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org," in *CNS*, 2013.
[4] R. Bonica, F. Baker, G. Huston, R. Hinden, O. Troan, and F. Gont, "IP Fragmentation Considered Fragile," RFC 8900, 2020.
[5] K. Fujiwara and P. Vixie, "IP Fragmentation Avoidance in DNS over UDP," RFC 9715, 2025.
[6] T. Wicinski, "DNS Privacy Considerations," RFC 9076, 2021.
[7] A. Hubert and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers," RFC 5452, 2009.
[8] D. Eastlake and M. Andrews, "Domain Name System (DNS) Cookies," RFC 7873, 2016.
[9] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)," RFC 7858, 2016.
[10] P. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," RFC 8484, 2018.
[11] C. Huitema, S. Dickinson, and A. Mankin, "DNS over Dedicated QUIC Connections," RFC 9250, 2022.
[12] T. V. Doan, I. Tsareva, and V. Bajpai, "Measuring DNS over TLS from the Edge: Adoption, Reliability, and Response Times," in *PAM*, 2021.
[13] R. Chhabra, P. Murley, D. Kumar, M. Bailey, and G. Wang, "Measuring DNS-over-HTTPS Performance Around the World," in *IMC*, 2021.
[14] M. Kosek, L. Schumann, R. Marx, T. V. Doan, and V. Bajpai, "DNS Privacy with Speed? Evaluating DNS over QUIC and its Impact on Web Performance," in *IMC*, 2022.
[15] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, 2021.
[16] M. Kühlewind and B. Trammell, "Applicability of the QUIC Transport Protocol," RFC 9308, 2022.
[17] P. Schwabe, D. Stebila, and T. Wiggers, "Post-Quantum TLS Without Handshake Signatures," in *CCS*, 2020.
[18] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698, 2012.
[19] A. S. Rawat and M. P. Jhanwar, "Quantum-safe Signatureless DNSSEC," in *AsiaCCS*, 2025.
[20] N. Weaver, C. Kreibich, B. Nechaev, and V. Paxson, "Implications of Netalyzr's DNS Measurements," in *SATIN*, 2011.
[21] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson, "Beyond the Radio: Illuminating the Higher Layers of Mobile Networks," in *MobiSys*, 2015.
[22] R. Bellis, "DNS Proxy Implementation Guidelines," RFC 5625, 2009.
[23] P. Mockapetris, "Domain Names—Concepts and Facilities," RFC 1034, 1987.
[24] ——, "Domain Names—Implementation and Specification," RFC 1035, 1987.
[25] J. Damas, M. Graff, and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))," RFC 6891, 2013.
[26] P. Hoffman, "DNS Security Extensions (DNSSEC)," RFC 9364, 2023.
[27] G. Alagic, E. Barker, L. Chen, D. Moody, A. Robinson, H. Silberg, and N. Waller, "Recommendations for Key-Encapsulation Mechanisms," NIST SP 800-227, 2025.
[28] D. Eastlake and A. Panitz, "Reserved Top Level DNS Names," RFC 2606, 1999.
[29] D. Eastlake, "Domain Name System (DNS) IANA Considerations," RFC 6895, 2013.
[30] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *CAV*, 2013.
[31] D. Dolev and A. C. Yao, "On the Security of Public Key Protocols," *IEEE Trans. Inf. Theory*, 1983.
[32] D. Stebila and M. Mosca, "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project," in *SAC*, 2017.
[33] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST SP 800-38D, 2007.
[34] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, 2010.
[35] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard," FIPS 203, 2024.
[36] ——, "Module-Lattice-Based Digital Signature Standard," FIPS 204, 2024.
[37] P.-A. Fouque *et al.*, "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU," Specification, 2020.
[38] National Institute of Standards and Technology, "Stateless Hash-Based Digital Signature Standard," FIPS 205, 2024.
[39] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, 2018.
[40] T. Pauly, E. Kinnear, C. A. Wood, P. McManus, and T. Jensen, "Discovery of Designated Resolvers," RFC 9462, 2023.
[41] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing," RFC 2827, 2000.
[42] H. Shulman, "Pretty Bad Privacy: Pitfalls of DNS Encryption," in *WPES*, 2014.
[43] S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, and C. Troncoso, "Encrypted DNS ⇒ Privacy? A Traffic Analysis Perspective," in *NDSS*, 2020.
[44] J. Bushart and C. Rossow, "Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS," in *FOCI*, 2020.
[45] S. Dickinson, B. Overeinder, R. van Rijswijk-Deij, and A. Mankin, "Recommendations for DNS Privacy Service Operators," RFC 8932, 2020.
[46] C. Lu, B. Liu, Z. Li, S. Hao, H. Duan, M. Zhang, C. Leng, Y. Liu, Z. Zhang, and J. Wu, "An End-to-End, Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come?" in *IMC*, 2019.
[47] A. Hounsel, K. Borgolte, P. Schmitt, J. Holland, and N. Feamster, "Comparing the Effects of DNS, DoT, and DoH on Web Performance," in *WWW*, 2020.
[48] T. Reddy, D. Wing, and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)," RFC 8094, 2017.
[49] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya, "Connection-Oriented DNS to Improve Privacy and Security," in *IEEE S&P*, 2015.
[50] P. Schmitt, A. Edmundson, A. Mankin, and N. Feamster, "Oblivious DNS: Practical Privacy for DNS Queries," *PoPETs*, 2019.
[51] E. Kinnear, P. McManus, T. Pauly, T. Verma, and C. A. Wood, "Oblivious DNS over HTTPS," RFC 9230, 2022.
[52] S. Bortzmeyer, R. Dolmans, and P. Hoffman, "DNS Query Name Minimisation to Improve Privacy," RFC 9156, 2021.
[53] R. Barnes, K. Bhargavan, B. Lipp, and C. A. Wood, "Hybrid Public Key Encryption," RFC 9180, 2022.
[54] D. J. Bernstein, T. Lange, J. Levin, and B.-Y. Yang, "PQConnect: Automated Post-Quantum End-to-End Tunnels," in *NDSS*, 2025.
[55] W. Wijngaards and G. Wiley, "Confidential DNS," Internet-Draft draft-wijngaards-dnsop-confidentialdns-03, 2015.
[56] P. Zuo *et al.*, "Approach on Encrypting DNS Message over UDP," Internet-Draft draft-zuo-dprive-encryption-over-udp-00, 2015.
[57] DNSCrypt Project, "DNSCrypt Protocol Specification Version 2," Protocol specification, 2013.
[58] F. Denis, "The DNSCrypt Protocol," Internet-Draft draft-denis-dprive-dnscrypt-11, work in progress, 2026.
[59] M. Dempsky, "DNSCurve: Link-Level Security for the Domain Name System," Internet-Draft draft-dempsky-dnscurve-01, 2010.
[60] D. J. Bernstein, "Internet: Preliminary Integration," PQCRYPTO Deliverable D2.2, 2018.
[61] D. Herrmann, K.-P. Fuchs, J. Lindemann, and H. Federrath, "EncDNS: A Lightweight Privacy-Preserving Name Resolution Service," in *ESORICS*, 2014.
[62] W. Kumari, E. Hunt, R. Arends, W. Hardaker, and D. Lawrence, "Extended DNS Errors," RFC 8914, 2020.

APPENDIX A
IMPLEMENTATION SPECIFICS

A. Validation Failure Handling

A malformed packet that otherwise identifies itself as VeilDNS is rejected rather than forwarded as ordinary DNS. Validation failures return `SERVFAIL` with an Extended DNS

TABLE XI
VeilDNS diagnostics. Strings omit the VEILDNS: prefix.

Diagnostic	Meaning
C2S_AEAD_FAIL	Client-to-server TXT payload fails AEAD authentication.
S2C_AEAD_FAIL	Server-to-client TXT payload fails AEAD authentication.
C2S_REPLAY	Client-to-server seq was already accepted.
S2C_REPLAY	Server-to-client seq was already accepted.
S2C_BINDING_FAIL	Authenticated response ID or stateful sequence does not match the query.
C2S_TXT_MISSING	Required client-to-server encrypted TXT record is missing.
S2C_TXT_MISSING	Required server-to-client encrypted TXT record is missing.
TXT_MALFORMED	TXT record does not match the expected encrypted format.
BAD_COOKIE	DNS Cookie is malformed, stale, or fails source-address validation.
BAD_KEM_ID	PRIVATE record names an unknown resolver KEM key identifier.
BAD_KEM	Required KEM ciphertext is missing or malformed for the selected mode.
CAPACITY	Resolver cannot admit new session state.
MALFORMED	Packet does not match the VeilDNS format.

TABLE XII
Key derivation and associated-data binding in VeilDNS.

Construction	Bound inputs
Static derivation inputs	"veildns", "v1", mode, suite, static KEM key identifier, KEM payload type 0x01, ct_s, sid_0, and sid_1.
Static key material	Static KEM shared secret ss_s.
Static master-key HKDF info	Static derivation inputs + direction ("c2s"/"s2c").
FS transcript hash	SHA-256 of the static derivation inputs + kid_{pk_e}, KEM payload type 0x02, pk_e, KEM payload type 0x01, and ct_e.
FS key material	$\mathsf{ss}_s \parallel \mathsf{ss}_e$.
FS master-key HKDF info	FS transcript hash + key phase ("provisional"/"final") + direction.
Packet key material	Directional master key.
Packet-key HKDF info	"veildns", "v1", mode, suite, direction, "packet", and packet context.
AEAD-AD	"veildns", "v1", mode, suite, direction, packet context, DNS header transaction ID, Flags, exchange-specific fields, and a copy of the OPT record with CLASS = 0.
FS exchange-1 response fields	Mode, suite, kid_{pk_e}, KEM payload type 0x02, and pk_e.

Error (EDE) option [62], except replayed queries in established state, which are silently dropped. EDE responses use code 0 (*Other Error*) with a diagnostic string as extra text. Table XI summarizes these diagnostics.

B. Cryptographic Encoding and Parameters

Table XII lists the bound inputs in order; each non-key input is prefixed by a two-byte big-endian length. Both sid_0 and sid_1 are encoded, using zero length when inapplicable.

HKDF-SHA256 uses veildns as its salt and the encoded inputs as its info. Static and FS directional derivations produce 32-byte master keys. Stateful packet contexts encode seq as an unsigned 64-bit big-endian integer. Per packet, HKDF expands the directional master key and packet context into a 32-byte AES key and 12-byte nonce for one-time use.

C. Miscellaneous

1) Replay Tracking: The 64-bit sequence number starts at zero, increases for each new query without wrapping, and is checked using a 64-entry sliding window per direction and key phase. A response uses the accepted query's sequence number. The window accepts unseen in-window sequences and rejects duplicate or older ones. For established traffic, the resolver authenticates a query and checks for replays before forwarding it to the CoreDNS backend.

2) Warmup Retries: Warmup responses are cached so that exact retries of stateful(-FS) exchange 1 or stateful-FS exchange 2 receive the same response. Cached responses do not outlive their associated session state. The exchange 2 response is stored with the final session state; if either is unavailable, the client restarts from exchange 1. If the first-response cache is absent, the resolver assigns a fresh sid_1 and derives a fresh response master key. After restart or eviction, the client can instead establish a new session with a fresh sid_0.

3) Pending Sessions: Stateful modes create resolver state before DNS Cookie validation. To mitigate resolver resource exhaustion from spoofed first-contact queries, the prototype limits this unvalidated state to 512 sessions, rejects new pending admissions at the limit, and expires entries after five seconds. A pending session becomes established only after a request with a valid Server Cookie and AEAD tag.

Appendix B
Additional Measurements

A. Lossy-Link Measurements

To evaluate performance under heavy loss, we repeated the local cold and warm experiments with 5% packet loss configured on the proxy-to-proxy path. For vanilla DNS and VeilDNS, we implement a simple recovery algorithm: after a timeout, the client proxy retries the query up to three times. For a one-way delay d, the timeout is $T_{\mathrm{retry}} = 2d{+}50\,\mathrm{ms}$. This gives 70 ms High-profile and 250 ms Low-profile timeouts. Ordinary retries in established state use a fresh seq. Exchange 1 and stateful-FS exchange 2 retries resend the original packet and receive the cached response.

The cold run used one batch of 500 sessions per mode, with fixed-load concurrency $c = 75$ for High and $c = 10$ for Low. The warm run used three independent 100-query batches per mode at $c = 5$ and reports the median values. The retry column reports N/A for DoX because TCP and QUIC retransmit internally. Tables XIII and XIV report the results.

B. Cloud Measurements

To observe client-facing latency on a routed Internet path, we repeated cold- and warm-phase experiments on two Ubuntu

24.04 LTS x64 DigitalOcean Regular CPU Droplets: a client-side proxy in SFO3 and a server-side proxy in NYC1, each with 4 vCPUs and 8 GB RAM. The measured inter-region RTT was approximately 67 ms.

We used the same protocol implementations as in the local testbed and ran both phases at fixed-load concurrency $c = 5$. The cold phase used 500 sessions per mode. The warm phase used three independent repetitions of 100 queries per mode and reports the median. Tables XV and XVI show the results.

TABLE XIII
COLD-PHASE BENCHMARK AT 5% PACKET LOSS. CONCURRENCY $c = 75$ FOR HIGH AND $c = 10$ FOR LOW; 500 COLD SESSIONS PER MODE.

		Outcome		Throughput	Latency (ms)		Bytes/session			Packets/session			CPU ms/session		
Profile	Mode	Success	Retries	Sessions/s	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
High	Vanilla DNS	500/500	57	1643.0	32.3	98.6	85.0	107.0	192.0	1.06	1.00	2.06	0.31	0.05	0.35
High	VeilDNS stateful/less	500/500	67	1451.8	34.0	98.3	1133.2	310.0	1443.2	1.07	1.00	2.07	0.35	0.12	0.47
High	VeilDNS stateful-FS	500/500	105	715.7	61.3	121.7	2257.4	1437.0	3694.4	2.11	2.00	4.11	0.62	0.25	0.87
High	DoT ML-DSA	500/500	N/A	146.1	229.9	1169.7	2041.2	8785.9	10827.1	13.87	13.08	26.95	0.89	1.09	1.98
High	DoH ML-DSA	499/500	N/A	134.3	225.2	1122.8	2371.6	9049.8	11421.4	16.58	15.79	32.38	0.99	1.24	2.23
High	DoQ ML-DSA	499/500	N/A	136.4	165.2	299.8	8152.1	10971.4	19123.6	14.68	13.80	28.48	1.34	1.82	3.16
Low	Vanilla DNS	500/500	62	40.7	240.6	467.0	84.5	107.0	191.5	1.06	1.00	2.06	0.87	0.15	1.01
Low	VeilDNS stateful/less	500/500	61	39.6	249.5	474.6	1125.2	312.4	1437.6	1.06	1.01	2.07	1.07	0.48	1.55
Low	VeilDNS stateful-FS	500/500	106	20.0	497.0	704.3	2264.6	1452.7	3717.3	2.12	2.01	4.13	2.20	1.05	3.25
Low	DoT ML-DSA	500/500	N/A	8.5	1158.8	2221.6	2060.5	8823.3	10883.7	14.34	13.79	28.14	4.26	5.30	9.56
Low	DoH ML-DSA	500/500	N/A	8.2	1205.4	2431.8	2468.2	9125.7	11593.9	18.55	17.11	35.66	4.44	5.40	9.84
Low	DoQ ML-DSA	496/500	N/A	6.4	1500.9	3065.0	8185.7	11162.6	19348.2	15.25	14.43	29.68	5.71	8.16	13.87

TABLE XIV
WARM-PHASE BENCHMARK AT 5% PACKET LOSS. CONCURRENCY $c = 5$; MEDIAN OF THREE 100-QUERY RUNS PER MODE.

		Outcome		Throughput	Latency (ms)		Bytes/query			Packets/query			CPU ms/query		
Profile	Mode	Success	Retries	QPS	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
High	Vanilla DNS	300/300	12	137.6	33.1	98.2	84.8	107.0	191.8	1.06	1.00	2.06	1.07	0.16	1.23
High	VeilDNS stateless	300/300	11	135.1	33.4	97.9	1143.7	310.0	1453.7	1.06	1.00	2.06	1.17	0.42	1.59
High	VeilDNS stateful(-FS)	300/300	11	155.2	31.7	97.3	305.8	310.0	615.8	1.04	1.00	2.04	1.12	0.31	1.44
High	DoT ML-DSA	300/300	N/A	146.8	33.7	57.4	135.4	164.8	300.6	1.35	1.37	2.75	0.82	0.69	1.51
High	DoH ML-DSA	300/300	N/A	111.1	44.1	84.2	210.4	205.0	414.2	1.35	1.32	2.67	1.18	0.95	2.12
High	DoQ ML-DSA	300/300	N/A	177.0	27.8	50.3	131.8	152.1	283.9	1.44	1.52	2.96	0.89	0.74	1.64
Low	Vanilla DNS	300/300	9	21.1	234.0	461.9	83.2	107.0	190.2	1.04	1.00	2.04	1.53	0.31	1.84
Low	VeilDNS stateless	300/300	10	20.1	244.0	469.1	1133.0	310.0	1443.0	1.05	1.00	2.05	1.79	0.76	2.55
Low	VeilDNS stateful(-FS)	300/300	13	20.1	245.4	469.6	314.6	310.0	624.6	1.07	1.00	2.07	1.60	0.55	2.15
Low	DoT ML-DSA	300/300	N/A	16.6	294.8	627.3	142.3	165.2	304.9	1.45	1.38	2.79	1.69	1.35	3.04
Low	DoH ML-DSA	300/300	N/A	16.7	295.6	475.4	207.3	209.4	419.3	1.37	1.41	2.81	1.48	1.35	2.83
Low	DoQ ML-DSA	300/300	N/A	20.6	234.3	423.6	138.8	159.1	297.9	1.55	1.64	3.19	2.02	1.60	3.62

TABLE XV
CLOUD COLD-PHASE BENCHMARK. SFO3 CLIENT TO NYC1 SERVER; RTT ≈ 67 MS; CONCURRENCY $c = 5$; 500 COLD SESSIONS PER MODE.

	Throughput	Latency (ms)		Bytes/session			Packets/session			CPU ms/session		
Mode	Sessions/s	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
Vanilla DNS	70.3	70.6	78.2	80	107	187	1.0	1.0	2.0	1.23	0.34	1.57
VeilDNS stateful/less	69.2	71.5	78.9	1063	310	1373	1.0	1.0	2.0	1.40	0.85	2.25
VeilDNS stateful-FS	34.7	143.0	155.2	2142	1437	3579	2.0	2.0	4.0	2.84	1.82	4.66
DoT ML-DSA	23.6	210.7	234.1	2044	8679	10724	14.1	13.0	27.1	4.69	5.06	9.75
DoH ML-DSA	23.5	211.7	234.7	2339	8921	11260	16.2	15.5	31.7	4.75	5.35	10.10
DoQ ML-DSA	17.7	280.1	311.1	7599	9685	17284	9.1	12.1	21.1	5.17	6.10	11.27

TABLE XVI
CLOUD WARM-PHASE BENCHMARK. SFO3 CLIENT TO NYC1 SERVER; RTT ≈ 67 MS; CONCURRENCY $c = 5$; MEDIAN OF THREE 100-QUERY RUNS.

	Throughput	Latency (ms)		Bytes/query			Packets/query			CPU ms/query		
Mode	QPS	Avg	p95	Sent	Recv	Total	Sent	Recv	Total	Client	Server	Total
Vanilla DNS	69.5	70.5	77.7	80	107	187	1.0	1.0	2.0	1.73	0.44	2.17
VeilDNS stateless	69.5	71.0	78.8	1079	310	1389	1.0	1.0	2.0	1.94	0.79	2.73
VeilDNS stateful(-FS)	68.2	71.1	78.3	294	310	604	1.0	1.0	2.0	1.87	0.80	2.67
DoT ML-DSA	66.4	74.6	82.3	136	157	293	1.4	1.3	2.7	2.99	1.36	4.35
DoH ML-DSA	68.5	72.7	76.1	206	202	407	1.4	1.3	2.7	1.79	1.35	3.14
DoQ ML-DSA	68.9	72.4	75.3	137	154	292	1.5	1.6	3.1	2.27	1.59	3.86