

Quantum-safe Signatureless DNSSEC

Aditya Singh Rawat
Ashoka University
aditya.rawat_phd21@ashoka.edu.in

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

Abstract—We present SL-DNSSEC: a backward-compatible protocol that leverages a quantum-safe KEM and a MAC to perform *signature-less* (SL) DNSSEC validations in a single UDP query/response style. Our experiments targeting NIST level I security for QTYPE A query resolution show that SL-DNSSEC is practically equivalent to the presently deployed RSA-2048 in terms of bandwidth usage and resolution speeds. Compared to post-quantum signatures, SL-DNSSEC reduces bandwidth consumption and resolution times by up to 95% and 60%, respectively. Moreover, with *response size* < *query size* ≤ 1232 bytes, SL-DNSSEC obviates the long-standing issues of IP fragmentation, TCP re-transmits and DDoS amplification attacks.

I. INTRODUCTION

A cryptanalytically relevant quantum computer (CRQC) running Shor’s algorithm [63] can efficiently solve the factoring and the discrete logarithm problem (DLP) in polynomial time. Asymmetric schemes, such as RSA and ECDSA, relying on the foregoing hardness assumptions, thus stand in urgent need to be replaced with their quantum-resilient counterparts. While a CRQC can also mount Grover’s [33] quadratically faster brute-force against symmetric algorithms (such as AES and SHA family), the urgency for a post-quantum transition in this case remains less pressing since a doubling of the key or hash size restores the original n -bit security.

Many modern protocols deployed over the Internet, such as TLS and SSH, rely on public-key cryptography 1) to provide message confidentiality and integrity, and 2) to authenticate the communicating participants. The DNS Security Extensions (DNSSEC) [57], [59], [58], being one among such protocols, facilitates the validation (origin authentication and data integrity) of DNS responses with the aid of digital signatures. Being the backbone of the Internet, the Domain Name System (DNS) maps a human-readable domain name (`www.example.com`) to a machine-understandable IP address (`1.2.3.4`). At present, DNS services are also utilized for email authentication [38], acquisition of TLS certificates by proving a domain’s ownership [9], and supporting Internet routing security (RPKI) [49], [20].

Without DNSSEC in place, DNS remains vulnerable to cache poisoning attacks [10], [11], [1] wherein an adversary can inject a false domain-to-IP mapping in a resolver’s cache, thereby eventually re-directing the users of the *poisoned* resolver to a malicious website. In order to perform a successful attack, an off-path adversary would need to simultaneously guess the 16-bit UDP source port and the 16-bit DNS transaction ID. However, recently researchers [50], [51] discovered critical vulnerabilities in DNS software stacks that narrowed this search space from 2^{32} to $2^{16} + 2^{16}$, effectively enabling them to compromise resolvers’ caches.

TABLE I: A size comparison (in bytes) of signature (sig) / ciphertext (ct) and public key (pk) of various algorithms.

Algorithm	Assumption	Quantum-safe	pk	ct / sig
X25519	ECDLP	✗	32	32
Kyber-512	Lattice	✓	800	768
ECDSA P-256	ECDLP	✗	64	64
RSA-2048	Factoring	✗	260	256
Falcon-512	Lattice	✓	897	666
Dilithium-2	Lattice	✓	1312	2420
SPHINCS ⁺ -128s	Hash	✓	32	7856

Although DNS over TLS (DoT) [36], DNS over HTTPS (DoH) [35], and DNS over QUIC (DoQ) [37] have been proposed, it is important to note that they are not a replacement for DNSSEC. The former, being privacy focused, establish an encrypted and authenticated channel between a client and a resolver (*i.e. hop-by-hop* security). On the other hand, DNSSEC 1) operates between resolvers and nameservers, and 2) guarantees the veracity of DNS records by establishing a chain of trust up to the root (*i.e. end-to-end* integrity).

In its endeavour to sustain Internet security in the face of quantum computers, the National Institute of Standards and Technology (NIST) has selected Crystals-Kyber [19] as Key Encapsulation Mechanism (KEM) and Crystals-Dilithium [28], Falcon [55] and SPHINCS⁺ [16] as digital signature algorithms. In comparison to their classical counterparts however, these algorithms (colloquially referred to under the umbrella acronym of PQC — designating Post-Quantum Cryptography), have significantly larger public key and signature / ciphertext sizes as elucidated in Table I above.

Size Constraints on DNS. With the quantum era on the horizon, DNSSEC must soon transition to CRQC-resistant algorithms so that it may continue to thwart cache poisoning attempts. However, the relatively larger footprint of PQC objects, as discussed above, will have major ramifications on the global DNS infrastructure. A DNS message, as originally specified, was restricted to a size of 512 bytes, with UDP being its primary transport. With a view to DNSSEC’s higher space requirements (for transferring signatures and public keys), this size bound was eventually increased to a *theoretical* value of 64 KB with Extension Mechanisms for DNS (EDNS0) [24]. Unfortunately, a DNS packet exceeding the Path MTU (Maximum Transmission Unit), which is usually 1500 bytes (<< 64 KB), triggers IP fragmentation at the intermediate routers. The ensuing UDP/IP fragments not only may never arrive [67], [18] (for *e.g.*, due to being blocked by stateless firewalls) but also can be used to exhaust a resolver’s resources [40] or to inject spoofed records in a DNS response [34]. Additionally, the study of [67] has shown that up to 10% of the resolvers fail to handle these fragments correctly.

In order to circumvent the multitude of issues linked with network layer fragmentation, DNS messages are recommended to not exceed **1232** bytes in size [3], [67], [54]. This conservative threshold, derived as 1280 (IPv6 minimum MTU) – 40 (IPv6 Header) – 8 (UDP Header), is deemed to prevent IP fragmentation on almost all network links [8], [53].

For conveying DNS messages that do not fit within the preceding size bracket, the proposed transport is TCP. In a usual DNS flow, when a response size exceeds the resolver’s advertised EDNS0 buffer (*i.e.* the maximum DNS message size it is willing to receive), a truncated response (with TC bit set in the header) is sent. Subsequently, the resolver discards the TC response (resulting in a wasted UDP trip) and retries the query (albeit with a different transaction ID) over TCP. Unfortunately, up to 11% of nameservers have been found to lack TCP support by [68], [54]. The report of [54] additionally remarks that TCP/53 connections could even be blocked by intruding middleboxes. In the surveys of [52], [26], a non-trivial number of resolvers did not properly fall back to TCP when requested by nameservers. Lastly, DNS over TCP has been shown to be measurably slower (sometimes by a factor of 4) and more resource intensive than DNS over UDP [5], [41], thus putting a limit on the number of TCP connections a DNS server might be able to handle concurrently.

Note that a properly implemented TCP support on nameservers and resolvers still does not clear away the road to a post-quantum DNSSEC. For *e.g.*, a DNS message containing just 3 SPHINCS⁺-256s level V signatures, a common scenario with non-minimal QTYPE A (IPv4 address) responses, even exceeds the maximum possible DNS message size of 64 KB.

A. Related Work

Out-of-Band Key Distribution. In [54], Müller *et al.* propose an out-of-band distribution (*i.e.* transportation outside the DNS infrastructure) of large public keys via HTTP or FTP. Unfortunately, not only does this approach require zone-operators to additionally maintain a web server, but it also has been shown to create a resolution overhead of about 30% in [13]. Furthermore, the size complications arising due to PQC signatures still remain largely unaddressed.

Merkle Tree Ladder (MTL). Fregly *et al.* [30] recently proposed a MTL mode which can reduce the size impact of PQC signature schemes. Specifically, the signer signs *Merkle tree ladders* that are derived from the messages to be validated. Individual messages are then authenticated relative to the ladder using a Merkle tree authentication path, while the ladder itself is validated using the public key.

Application Layer Fragmentation. With an aim of avoiding the fragility and the unavailability connected with IP fragmentation and TCP fallbacks, respectively, many proposals have been put forward that fragment large DNS messages at the application (DNS) layer. In such a scenario, the nameserver becomes responsible for the fragmentation of a DNS response and the resolver for the subsequent reassembly thereof.

Sivaraman *et al.* [64] fragmented a large DNS response across multiple UDP datagrams, transmitting each fragment sequentially. On the other hand, ATR [65] (though not strictly a fragmentation scheme) sent an additional TC response to

trigger an immediate TCP fallback on the client, in case the original large response failed to arrive. Unfortunately, both of these proposals failed to gain traction since multiple responses were being sent out to a single request. Many firewalls are configured with the policy of accepting one response packet per query. Moreover, many resolvers close their sockets immediately after receiving the first response packet. Thus, there were concerns about ICMP flooding since for each trailing response packet that could not be delivered, a *destination unreachable* packet would be sent back to the nameserver.

Addressing the shortcomings of the previous drafts, ARRF [32] fragments DNS resource records and sends an additional response only upon an explicit *request*. Since each extra response has its own query, prior concerns about firewalls and ICMP flooding are mitigated. Unfortunately, ARRF fragments, owing to their use of non-standard Type RRRFRAG pseudo-records, could be potentially dropped by inspecting middleboxes. Secondly, ARRF remains vulnerable to memory exhaustion attacks, as acknowledged by its authors in [32]. Finally, ARRF requires a minimum of two round-trips to reconstruct the full DNS message.

A recent work, called QNAME-Based Fragmentation (QBF) [56], achieves a one round-trip reassembly of post-quantum DNSSEC messages while using only standard DNS record Type(s). Unlike previous schemes, it fragments raw signature and public key bytes stored in RRSIG and DNSKEY records, respectively. The implication is that the fragments *resemble* the original DNS response, except insofar as they carry partial signatures / public keys. A fragment is explicitly *requested* by encoding the desired fragment number in the QNAME field of a query. Lastly, QBF is backwards-compatible and not susceptible to memory-depletion attacks.

Discussion on ARRF / QBF. For fast query resolutions, both ARRF (in 2nd round trip) and QBF (in 1st round trip) send multiple DNS over UDP messages in parallel. On busy resolvers and nameservers, handling thousands of queries per second, this deluge of DNS packets could lead to a starvation of network bandwidth. Moreover, such bursts in traffic can conceivably overwhelm load balancers or trigger flood protection in firewalls. This is because unlike its TCP sibling, UDP does not have any built-in flow and congestion control mechanisms.

Furthermore, it is crucial to remember that UDP/IP does not guarantee a reliable delivery of packets. In ARRF/QBF, as the number of signatures to transmit or the sizes thereof grow (from configuring higher NIST levels), the number of DNS messages that need to be exchanged also inevitably rises. Therefore, the probability of at least one DNS query/response packet getting dropped during transit also increases, resulting in unforeseen resolution delays or timeouts.

To give a perspective, given a 1% network loss rate and SPHINCS⁺-128s as the zone signing algorithm, the probability of at least one QBF packet being lost during transit can be calculated as $\Pr = 1 - (0.99)^{46} = 0.37$, where 46 is the total number of DNS packets exchanged during the session. This implies that, with a one-third probability, a QBF SPHINCS⁺ session will require an extra round-trip. While the picture is not as bleak with Falcon and Dilithium, it is circumspect to be prepared for all circumstances, especially since SPHINCS⁺ still remains the most conservative choice among its siblings.

Another concern with ARRF/QBF is their potential to be exploited for a DDoS attack [39], [69], [60], wherein small DNS over UDP queries with a spoofed source IP address cause large DNS responses (amplification) to be sent out from a server to a target IP device (reflection), eventually overwhelming the latter or the network thereof. In one of the major DDoS events, the attackers were able to generate 300 Gbps of traffic on a Tier 1 provider using open DNS resolvers [2]. On a related note, performing such type of attacks over TCP is not feasible because of the three-way TCP handshake. This is because client’s query is forwarded to the DNS software only after receiving a valid¹ client ACK to the server SYN.

Bearing the above apprehensions in mind, it appears that fragmentation schemes may not be the panacea for DNSSEC’s complications in the quantum age. Therefore, in this work, we take a fundamentally different approach by performing DNSSEC validations without PQC signatures. More precisely, we leverage the concept of authentication via a key exchange.

Authenticated Key Exchange without Signatures. The notion of an authenticated key exchange (AKE) follows a long succession of works, with the very early proposals being [15], [21]. In the SKEME protocol of Krawczyk [44] and the RSA key-transport (in TLS versions up to 1.2), an entity is authenticated via a successful decryption of a challenge message. The protocol of Bellare *et al.* [14] obtained authentication from long-term Diffie-Hellman (DH) keys. In particular, the resulting shared secret is fed into the session key calculation to derive an *implicitly* authenticated key (*i.e.* only the legitimate parties could compute it). Other DH-based AKE protocols include MQV [48], HMQV [45], NAXOS [47], Noise [6], Signal [4] and WireGuard [27]. Constructions using generic Key Encapsulation Mechanisms (KEMs) for AKE have also been proposed in [25], [31].

In the domain of TLS 1.3, the OPTLS proposal [46] is a DH-based AKE that offers a signature-free handshake. Specifically, the server sends a certificate containing a DH public key whilst combining the corresponding long-term secret key with the ephemeral public key from the client. The resulting shared key is then used to generate a MAC which authenticates the server. Unfortunately, there does not yet exist an efficient OPTLS instantiation for a post-quantum setting ([70], Ch. 12). KEMTLS [61], which builds upon OPTLS, is a KEM-based AKE that bypasses the usual signed-DH flow of TLS to achieve a signature-less PQC handshake. More concretely, the client performs an encapsulation against the server’s KEM public key (obtained via the `ServerCertificate` message during the handshake) to derive an *implicitly* authenticated shared secret, which is then used to encrypt the first flight of application data from the client. The server is later *explicitly* authenticated with the `ServerFinished` message. Note that to validate the server’s KEM public key, the client still unavoidably relies upon a CA signature. A follow-up work by the same authors, called KEMTLS-PDK [62], is a variant of KEMTLS that uses pre-distributed keys for earlier authentication. This scenario occurs when a web-browser caches certificates or in the case of IoT (Internet of Things) devices or mobile applications that come with pre-bundled certificates.

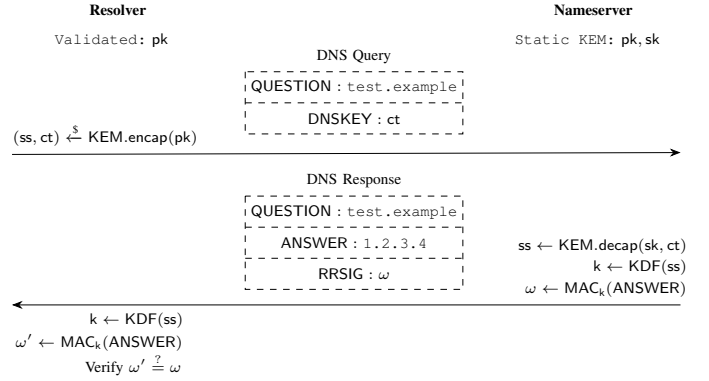


Fig. 1: An abstracted view of SL-DNSSEC validating the answer IP 1.2.3.4. The resolver has already fetched and DNSSEC-validated the KEM public key of the nameserver.

B. Our Contributions

Given the practical size constraint on DNS(SEC) messages that impedes a smooth adoption of post-quantum cryptography, we illustrate how an authenticated key exchange (AKE) can be used to achieve a signature-free validation of DNS resource records. To this end, we propose SL-DNSSEC: an AKE-based protocol for DNSSEC which uses 1) A quantum-safe KEM to first establish a shared key between a resolver and a nameserver, and 2) a Message Authentication Code (MAC), computed under the shared key, to simultaneously authenticate a DNS record’s origin and verify its integrity.

An overview of the protocol is illustrated in Figure 1. The resolver holds a DNSSEC-validated KEM public key of the nameserver. Using a series of KEM and KDF (Key Derivation Function) operations, both parties derive a symmetric MAC key. The nameserver sends a MAC authentication tag instead of a digital signature on the answer record.

We now outline the salient benefits of SL-DNSSEC with a summary thereof in Table II. All numerical values have been inferred from Tables (XI, XII) in §V-B1 of this paper.

- **Massive bandwidth savings.** Compared to SPHINCS⁺, Dilithium and Falcon, SL-DNSSEC transfers about 95%, 86%, and 58% less data (Fig. 2) during a typical QTYPE A query resolution.
- **Fast 1-RTT resolution.** SL-DNSSEC remains 50% – 60% faster than DNSSEC over Standard DNS (SD), with the latter incurring the speed penalty of a wasted UDP trip and then of a three-way TCP handshake.
- **1 packet sent/received.** Although ARRF/QBF take 2 and 1 round-trip(s) (respectively), they exchange multiple packets (up to 46) in parallel, thereby increasing the chances of packet drops and UDP flooding. SL-DNSSEC, however, sends a single query/response.
- **DDoS mitigation.** With *smaller* responses than queries (Fig. 3), SL-DNSSEC fulfils the takeaway 6 in the vision paper [39] of being an “*amplification-resistant solution for post-quantum DNSSEC*”.

¹With Acknowledgment Number = Server Sequence Number + 1

Total bandwidth usage

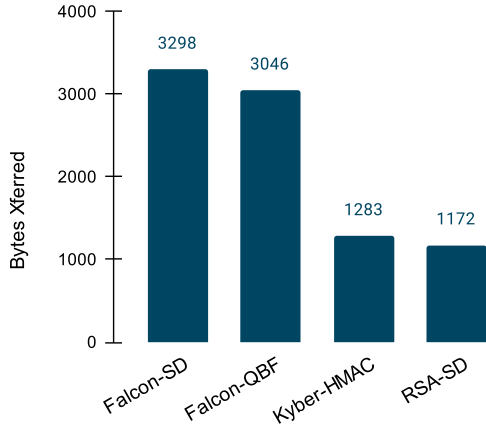


Fig. 2: A total bandwidth usage comparison between Kyber-HMAC (SL-DNSSEC) and signature-based DNSSEC methods. SD denotes Standard DNS.

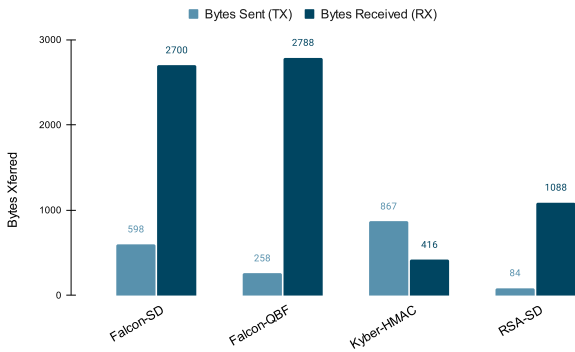


Fig. 3: A transmit (TX) / receive (RX) bandwidth usage comparison between Kyber-HMAC (SL-DNSSEC) and signature-based DNSSEC methods. SD denotes Standard DNS.

- **Backward compatibility.** SL-DNSSEC uses standard record Type(s) and wire format to ensure that messages pass through stringent firewalls. It also allows for a graceful fallback to the regular DNSSEC flow should one of the endpoints happen to be protocol-oblivious. Moreover, a zone can deploy SL-DNSSEC without requiring its parent to be protocol-aware.

To evaluate SL-DNSSEC, we program a daemon that can run atop any DNS provider (such as BIND or PowerDNS). The daemon performs all SL-DNSSEC-related operations on behalf of the DNS software. In fact, no changes to the underlying DNS stack are required, except for a small patch on the resolver’s side to detect the Z-bit in the HEADER.

Availability. The software artifact germane to this work is available at: <https://github.com/b8b1/sl-dnssec>.

TABLE II: A comparison between SL-DNSSEC and signature-based DNSSEC methods. SD : Standard DNS (TCP fallback).

	SL-DNSSEC	DNSSEC over SD	DNSSEC over ARRF / QBF
No TCP fallback	✓	✗	✓
Low bandwidth usage	✓	✗	✗
Fast resolution	✓	✗	✓
DDoS amp. resistant	✓	✓	✗
No network flooding	✓	✓	✗
1 packet sent/recvd.	✓	✗	✗
Reliability	✓	✓	✗

II. PRELIMINARIES

Notations. The term *resource record* (RR) is often referred to as simply a *record*. $||$ represents concatenation. $X \rightarrow Y$ denotes member Y of an abstract structure X. In the context of networking protocols, A/B indicates A over B (For *e.g.*, DNS/UDP — DNS over UDP). RTT stands for round-trip time. ANS is short for Authoritative Name Server. For presentation, we omit the root label (*i.e.* the trailing period $.$) as in *example.com.*) while writing fully qualified domain names (FQDNs). The word *transfer* is occasionally abbreviated as *xfer*. The Bandwidth Amplification Factor (BAF) of a DNS over UDP session is calculated as below:

$$\text{BAF} = \frac{\text{Number of bytes received (RX)}}{\text{Number of bytes sent (TX)}}$$

A. Domain Name System (DNS)

We briefly review the relevant background on DNS. Consider a canonical domain name: *www.example.com.* (with the trailing dot). Each label: (*www*), (*example*), (*com*) and ($.$)² corresponds to a level within the DNS hierarchy, with the root ($.$) being at the apex. Under the root come *top-level domains* or TLDs (*com*), and within these are *second-level domains* (*example*), and then *subdomains* (*www*). A name-server that contains definitive information for the zone is said to be *authoritative* for the zone. For *e.g.*, *example.com* ANS is authoritative over the A record for *www.example.com*.

DNS Lookup. To retrieve the IP address of *www.example.com*, the client (*stub resolver*) sends a *recursive* QTYPE A DNS query to its resolver (local DNS server). The resolver, in the event of not having the answer in its cache, performs the following steps *iteratively*:

- 1) It sends a QTYPE NS query to a root ($.$) name-server, which subsequently responds with the following *glue* (referral) records: 1) A Type NS record containing the domain name of *com* nameserver 2) A Type A record containing the IP of *com* nameserver.
- 2) It sends a QTYPE NS query to the *com* nameserver, which then responds with the following *glue* records: 1) A Type NS record containing the domain name of *example.com* nameserver 2) A Type A record containing the IP of *example.com* nameserver.

²The root label is technically null.

TABLE III: DNS HEADER Wire Format

ID															
QR	OpCode		AA	TC	RD	RA	Z	AD	CD	RCode					
QDCOUNT															
ANCOUNT															
NSCOUNT															
ARCOUNT															

- ID: used by requester to match a response to its query
- QR: whether message is a query (0) or a response (1)
- AA: whether response is authoritative (1) or not (0)
- TC: whether response is truncated (1) or not (0)
- Z: a reserved bit set to 0 by default
- AD: whether response has authenticated data (1) or not (0)
- RCode: (0) NOERROR; (1) FORMERR – query format error; (2) SERVFAIL – server failure; (3) NXDOMAIN – domain name does not exist

- 3) It sends a QTYPE A query to the `example.com` nameserver, which finally responds with a Type A record containing the IP of `www.example.com`.
- 4) It caches and forwards the received IP to the client.

DNS Wire Format. A generic DNS message is divided into 5 sections: HEADER, Question, Answer, Authority, and Additional. The HEADER is always present and has a constant size of 12 bytes. Table III presents the wire format of a DNS HEADER. The Question section consists of the following fields: QNAME (specifies the domain name encoded in the standard DNS name notation. For *e.g.*, `test.example` is encoded as `[4]test[7]example[0]`), QTYPE (specifies the type of DNS records being requested), and QCLASS (specifies the class of the query, by default set to `IN` *i.e.* Internet). The last three sections have the same format: a possibly empty list of concatenated DNS records.

The DNS resource records (RRs) are database entries that provide information about a domain name. Each record has the following sections: NAME (specifies the domain name encoded in standard notation), TYPE (indicates the type of RR), CLASS (specifies the class of data, defaults to `IN`), TTL (time-to-live in seconds *i.e.* how long the RR can stay cached), RDLENGTH (specifies the length in bytes of the RDATA field), and RDATA (contains the actual data associated with the record). The Type A and AAAA records contain IPv4 and IPv6 addresses in their RDATA fields, respectively. The Answer section contains records that answer the question; the Authority section contains records that point toward an ANS; the Additional section contains records which relate to the query, but are not strictly answers to the question.

OPT Record. EDNS0 [24] introduces a pseudo-record called OPT (short for *options*) in the Additional section of a DNS message. Note that unlike traditional resource records, pseudo-records do not actually exist in a zone file and are instead created on-the-fly. In queries, a requester specifies the maximum DNS message size it is willing to accept (also known as EDNS0 buffer size or UDP payload size) in `OPT → CLASS`. In addition to this, the requester also indicates its ability to handle DNSSEC records by setting the `DO` (DNSSEC OK) bit in `OPT → TTL`.

TABLE IV: RRSIG Wire Format

RRSIG Record					
NAME	TYPE = RRSIG	CLASS	TTL	RDLENGTH	
RDATA					
Type Covered	Type of records signed				
Algorithm	Signature algorithm used				
Labels	Number of labels in the signed name				
Original TTL	Original time-to-live of the records signed				
Signature Expiration	When the signature expires				
Signature Inception	When the records were signed				
Key Tag	ID of the key to be used for signature verification				
Signer's Name	Name of the signer				
Signature	← <code>sign(RRSIG → RDATA RR(1) RR(2) ...)</code> where RDATA excludes Signature and <code>RR(i)</code> is the <i>i</i> -th record in the RRset				

TABLE V: DNSKEY Wire Format

DNSKEY Record					
NAME	TYPE = DNSKEY	CLASS	TTL	RDLENGTH	
RDATA					
Flags	Specifies whether the key is a ZSK (256) or a KSK (257)				
Protocol	Always set to <code>0x03</code> to indicate DNSSEC				
Algorithm	Signature algorithm of the key				
Public Key	Contains the raw public key bytes				

`OPT → RDATA` contains DNS cookies [29] which provide a limited security against certain off-path attacks such as denial-of-service, cache poisoning, and answer forgery.

B. DNS Security Extensions (DNSSEC)

DNSSEC enhances the security of DNS by ensuring the authenticity and integrity of resource records. To realize this aim, it introduces three³ new types of resource records: Resource Record Signature (RRSIG), DNS Public Key (DNSKEY), and Delegation Signer (DS).

1) RRSIG. A digital signature is computed, using a secret key (discussed below) over a set (called an RRset) of DNS resource records that have the same NAME, CLASS and TYPE. The resulting signature is stored in the RDATA → Signature field of an RRSIG record (Table IV).

2) DNSKEY. A DNSKEY record (Table V) stores a public key. Each zone employs two types of keys: Zone Signing Key (ZSK) and Key Signing Key (KSK). KSK is used to sign only DNSKEY RRsets while ZSK is used to sign all other RRsets. Whenever a resolver receives a DNS response with an RRSIG record, it uses the associated DNSKEY record to verify the digital signature contained therein.

3) Delegation Signer (DS). The DS record (Table VI) plays a pivotal role in *recursively* constructing a secure chain of trust from a child zone to the DNS root (.).

Whenever a resolver verifies RRSIGs using the ZSK_{pk} of a child zone, it must also ascertain the authenticity of that key. Recall that the DNSKEY RRset containing ZSK_{pk} and KSK_{pk} is signed using the child's KSK_{sk} . Since KSK is ultimately self-signed, a resolver must also connect the trust thereof with the child's parent. To specifically aid resolvers in this endeavour, the child generates a cryptographic hash of its KSK_{pk} and shares it with its parent in a DS record.

³A fourth Type NSEC(3) record, used to verify the non-existence of a record name and type, is outside the purview of this work.

TABLE VI: DS Wire Format

DS Record	
NAME	TYPE = DS CLASS TTL RLENGTH
RDATA	
Key Tag	ID of the KSK which is hashed
Algorithm	Signature algorithm of the key
Digest Type	Hash algorithm
Digest	$\leftarrow \text{hash}(\text{DNSKEY} \rightarrow \text{NAME} \parallel \text{DNSKEY} \rightarrow \text{RDATA})$

During a DNS lookup, when a resolver is referred to a child zone by its parent, the latter provides a DS record containing the hash of the child’s KSK_{pk} . This DS record is what indicates to the resolver that the child zone is DNSSEC-enabled. More importantly, the parent also furnishes an RRSIG on this DS record using its own ZSK_{sk} .

To validate the child zone’s KSK_{pk} , the resolver hashes it and compares it to the DS record from the parent. Additionally, the resolver also verifies the associated RRSIG of that DS record using the ZSK_{pk} of the parent.

DNSSEC Lookup. This is similar to the DNS lookup described in §II-A, except that the resolver now sets the DO (DNSSEC OK) bit in its query. The following extra records are therefore returned at each step:

- 1) The root (.) nameserver also sends com’s DS and RRSIG thereon created with (.)’s ZSK_{sk} . Additionally, it sends (on an explicit QTYPE DNSKEY query) (.)’s DNSKEYs and RRSIG thereon created with (.)’s KSK_{sk} . Here, we assume the resolver already holds (.)’s KSK_{pk} as the *trust anchor*.
- 2) The com nameserver also sends example.com’s DS and RRSIG thereon created with com’s ZSK_{sk} . Additionally, it sends (on an explicit QTYPE DNSKEY query) com’s DNSKEYs and RRSIG thereon created with com’s KSK_{sk} .
- 3) The example.com nameserver also sends RRSIG created with its ZSK_{sk} on the Type A record containing the answer IP. Additionally, it sends (on an explicit QTYPE DNSKEY query) its DNSKEYs and RRSIG thereon created with its KSK_{sk} .

On a successful DNSSEC validation, the resolver sends its answer response to the client with HEADER \rightarrow AD set.

C. Key Encapsulation Mechanism (KEM)

Definition 1: A Key Encapsulation Mechanism (KEM) is an asymmetric primitive that allows two parties to establish a shared secret in a key space \mathcal{K} .

A KEM instance defines three probabilistic operations:

- **Key Generation:** $\text{KEM.keygen}()$ generates a public and private keypair (pk, sk) .
- **Encapsulation:** $\text{KEM.encap}(pk)$ generates a shared secret ss in a key space \mathcal{K} and a ciphertext (encapsulation) ct against pk .
- **Decapsulation:** $\text{KEM.decap}(sk, ct)$ takes as input sk and ct , and decapsulates the shared secret $ss' \in \mathcal{K}$. In a δ -correct scheme, $\Pr(ss = ss') \geq 1 - \delta$.

Security Model. Shared secret (ss) should be indistinguishable from random (IND), given just pk (Chosen Plaintext Attack (CPA)) or additionally given access to a decapsulation oracle (Chosen Ciphertext Attack (CCA)).

III. THE SL-DNSSEC PROTOCOL

SL-DNSSEC is a backward-compatible and amplification-resistant protocol for DNSSEC that validates DNS resource records without signatures. Pursuant to this objective, it uses as its primary building blocks: 1) A post-quantum KEM to first establish a shared secret between a resolver and a nameserver 2) A KDF to derive a symmetric MAC key of an appropriate length from the shared secret, and 3) A MAC to compute authentication tags on DNS records.

On a high-level, a nameserver generates a KEM keypair and adds the public key to its DNSKEY RRset, re-signing the latter with its KSK_{sk} . This RRset is fetched and DNSSEC-validated by a resolver using the covering RRSIG and a signed DS record from the zone’s parent. For any subsequent interaction, the resolver and the nameserver compute a symmetric MAC key using KEM and KDF operations. The nameserver then sends MAC tags instead of signatures on DNS RRsets.

We now demonstrate the execution of the SL-DNSSEC protocol between a resolver and an ANS. Note that SL-DNSSEC can also be deployed on other zones, such as (.) or com.

The protocol is broadly divided into four phases.

A. Phase 1: KEM Key Generation

Assume a DNSSEC-enabled zone (say, example.com) with a Key Signing Key ($\text{KSK}_{pk}, \text{KSK}_{sk}$) and a Zone Signing Key ($\text{ZSK}_{pk}, \text{ZSK}_{sk}$). Therefore, the current public key RRset of example.com comprises: 2 Type DNSKEY records containing KSK_{pk} and ZSK_{pk} , respectively.

The zone operator now runs $\text{KEM.keygen}()$ to generate a *Zone KEM Key*: $(\text{ZKK}_{pk}, \text{ZKK}_{sk})$.

Thereafter, the operator performs the following steps:

- 1) Create a generic DNSKEY record.
- 2) Set $\text{DNSKEY} \rightarrow \text{RDATA} \rightarrow \text{Flags} = 258$
- 3) Set $\text{DNSKEY} \rightarrow \text{RDATA} \rightarrow \text{Algorithm} = \text{KEM}$
- 4) Set $\text{DNSKEY} \rightarrow \text{RDATA} \rightarrow \text{PublicKey} = \text{ZKK}_{pk}$
- 5) Add DNSKEY to the existing RRset of public keys.
- 6) Re-sign the RRset using KSK_{sk} .

Here, the value 258 for Flags is one of the available choices after turning off the SEP (secure entry point) bit (refer RFC [59] §2.1.1). Note that the SEP flag is set only for a KSK which has a DS record in the parent zone. Furthermore, in some DNS software, the signature over a public key RRset is computed using both KSK_{sk} and ZSK_{sk} , thus resulting in two RRSIGs (consult [23], §4.7).

When a resolver now sends a QTYPE DNSKEY query to example.com, it will receive a DNS response (consult Table VII for its structure) containing the following records:

- Three DNSKEY records holding KSK_{pk} , ZSK_{pk} and ZKK_{pk} , together constituting 1 RRset
- One covering RRSIG using KSK_{sk} on the RRset

TABLE VII: An abstracted view of a QTYPE DNSKEY response containing the public keys of `example.com` zone

```

-----
Header Section
Question Section
QNAME = example.com
QTYPE = DNSKEY
QCLASS = IN
Answer Section
DNSKEY KSKpk
DNSKEY ZSKpk
DNSKEY ZKKpk
RRSIG with KSKsk
Authority Section
Additional Section
OPT
-----

```

The implication of the RRSIG is that the trust of the KEM key (ZKK) can now be established with the zone’s parent and then recursively with the root. This is easy to see since the `com` zone (the parent of `example.com`) already holds a DS record bearing the hash of KSK_{pk} .

The resolver thereupon verifies the RRSIG using KSK_{pk} and then validates KSK_{pk} itself via the signed DS record it had earlier received from the `com` nameserver during the referral.

Finally, observe that the `com` zone and the root are not required to be SL-DNSSEC-aware during the entire phase.

B. Phase 2: Preparing a SL-DNSSEC Query

Assume that a resolver intends to send a QTYPE A query with a QNAME `www.example.com` to the `example.com` ANS. We additionally presume that the resolver has already fetched and validated the QTYPE DNSKEY response (as outlined in §III-A) from the ANS.

The aforesaid is a common scenario in DNS, wherein a resolver already stores the DNSKEYs of previously contacted zones in its cache. Alternatively, we suggest that resolvers retrieve the public keys of an unacquainted zone first, before dispatching their main query.

The resolver now executes the following operations:

- 1) Create a QTYPE A query message, say Q.
- 2) Perform a KEM encapsulation against ZKK_{pk} of `example.com` to probabilistically obtain a shared secret ss and ciphertext ct .

$$(ss, ct) \xleftarrow{\$} \text{KEM.encap}(ZKK_{pk})$$

- 3) Create a generic DNSKEY record.
- 4) Set DNSKEY \rightarrow RDATA \rightarrow Flags = ZKK_{ID}
- 5) Set DNSKEY \rightarrow RDATA \rightarrow Algorithm = KEM
- 6) Set DNSKEY \rightarrow RDATA \rightarrow Public Key = ct
- 7) Insert DNSKEY in Q \rightarrow Additional section.
- 8) Send Q.

In settings wherein a zone offers multiple KEM public keys, a resolver additionally needs to include information about the particular KEM key it has used for encapsulation. To this end, we repurpose the 2-byte `Flags` field. Specifically, the resolver computes the 2-byte `Key Tag`, say ZKK_{ID} , using the Type DNSKEY record of the utilized KEM key (refer RFC [59], Appendix B for the algorithm used for `Key Tag` computation) and sets `Flags = ZKKID` as previously outlined.

TABLE VIII: Wire format: SL-DNSSEC Query Q containing the KEM ciphertext ct

```

-----
Header Section
Question Section
QNAME = www.example.com
QTYPE = A
QCLASS = IN
Answer Section
Authority Section
Additional Section
NAME = example.com
TYPE = DNSKEY
:
:
RDLENGTH = x
RDATA
Flags = ZKKID
Protocol = 0x03
Algorithm = KEM
Public Key = 0x2a4b... (ct)
OPT
-----

```

TABLE IX: An abstracted view of a non-minimal DNS response to a QTYPE A query

```

-----
Header Section
Question Section
QNAME = www.example.com
QTYPE = A
QCLASS = IN
Answer Section
RR1 TYPE A
RRSIG1
Authority Section
RR2 TYPE NS
RRSIG2
Additional Section
RR3 TYPE A
RRSIG3
OPT
-----

```

An example wire format of the resulting DNS query Q is illustrated in Table VIII. Notice that Q transports ciphertext ct using standard DNS records and wire format. Moreover, inserting DNSKEY record (bearing ct) in the Additional section further improves backward compatibility, since a SL-DNSSEC-oblivious ANS would ignore it and proceed with the usual flow.

C. Phase 3: Preparing a SL-DNSSEC Response

On receiving the DNS query Q, the `example.com` ANS executes the following actions:

- 1) Prepare a traditional QTYPE A DNS response (say, R) to Q. In this example, we assume R to be a *non-minimal* DNS response (refer Table IX for its general format) which contains the following records⁴:
 - a) 1 Type A resource record (RR_1) in Answer section containing the answer IP address and 1 covering $RRSIG_1$
 - b) 1 Type NS resource record (RR_2) in Authoritative section containing the nameserver’s name and 1 covering $RRSIG_2$
 - c) 1 Type A resource record (RR_3) in Additional section containing the nameserver’s IP address and 1 covering $RRSIG_3$
- 2) Check if the size of R is within:
 - a) Resolver’s UDP *receive* limit, as publicized in Q \rightarrow OPT \rightarrow CLASS
 - b) Nameserver’s own UDP *send* limit, as configured in `named.conf`

⁴For simplicity, here each RRset contains only 1 resource record.

— If affirmative, the ANS has the option to send *R as it is* (*i.e.* with signatures). In this example, we presume this check to return negative (which is expected with PQC signatures).

- 3) Check for a DNSKEY record containing a KEM ciphertext in $Q \rightarrow$ Additional section.
 - If negative, continue with the regular DNSSEC flow. Otherwise, proceed as below.
- 4) Extract the ciphertext ct from DNSKEY record and perform a KEM decapsulation using ZKK_{sk} to obtain the shared secret ss .

$$ss \leftarrow \text{KEM.decaps}(ZKK_{sk}, ct)$$

— In case the ANS holds multiple KEM ZKCs, the correct key for the decapsulation can be identified using the Key Tag (ZKK_{ID}) provided by the resolver in the Flags field (see §III-B).

- 5) Feed ss to a secure KDF to derive a key k of the requisite length.

$$k \leftarrow \text{KDF}(ss)$$

- 6) For every RRSIG_i in response R , do:
 - a) Set $\text{RRSIG}_i \rightarrow \text{RDATA} \rightarrow \text{Alg.} = \text{KEM}$
 - b) Set $\text{RRSIG}_i \rightarrow \text{RDATA} \rightarrow \text{Key Tag} = ZKK_{ID}$
 - c) Let $\text{msg} := \text{RRSIG}_i \rightarrow \text{RDATA} \parallel \text{RR}_i(1) \parallel \dots$
 - where RDATA excludes Signature and $\text{RR}_i(j)$ is the j -th record in RRset_i
 - d) Compute $\omega \leftarrow \text{MAC}_k(\text{msg})$
 - e) Set $\text{RRSIG}_i \rightarrow \text{RDATA} \rightarrow \text{Signature} = \omega$
- 7) Send R .

Table X depicts a comparison between the original DNS response containing PQC signatures (here, Falcon) and the SL-DNSSEC response containing MACs. Note that $\text{RDLENGTH } y \ll \text{RDLENGTH } x$ since MACs are usually much *smaller* than post-quantum signatures. The implication here is that as the number of RRSIG s increase, the size disparity between a signature-based response and its SL-DNSSEC counterpart becomes even more exaggerated.

A noteworthy distinction between SL-DNSSEC and the conventional DNSSEC flow is also herein encountered. While in the latter approach, signatures are usually pre-generated (*i.e.* the zone file is signed offline and then published on the nameserver), the former computes MACs *on-the-fly*.

Finally, observe that a MAC is computed over exactly the same message as that specified for a signature in the RFC (refer §3.1.8.1. in [59]). Additionally, the response R , like the query Q , uses standard record types and wire format.

D. Phase 4: Validating a SL-DNSSEC Response

In due course, when the resolver receives the DNS response R , it proceeds to validate the resource records contained therein in the following manner:

- 1) Fetch the shared secret ss from the state.
- 2) Feed ss to the KDF to derive the key k .

$$k \leftarrow \text{KDF}(ss)$$

TABLE X: Wire format: Original response with signatures (Left), SL-DNSSEC response with MACs (Right)

Original Response with Signatures (Left)	SL-DNSSEC Response with MACs (Right)
<pre> Header Section Question Section QNAME = www.example.com QTYPE = A QCLASS = IN Answer Section NAME = www.example.com TYPE = A : RDLENGTH = 4 RDATA = 1.2.3.4 NAME = www.example.com TYPE = RRSIG : RDLENGTH = x RDATA Type Covered = A Algorithm = FALCON : Key Tag = ZSK_ID Signer's Name = example.com Signature = 0x1a2b...(σ1) Authority Section : Signature = 0x3c4d...(σ2) Additional Section : Signature = 0x5e6f...(σ3) OPT </pre>	<pre> Header Section Question Section QNAME = www.example.com QTYPE = A QCLASS = IN Answer Section NAME = www.example.com TYPE = A : RDLENGTH = 4 RDATA = 1.2.3.4 NAME = www.example.com TYPE = RRSIG : RDLENGTH = y RDATA Type Covered = A Algorithm = KEM : Key Tag = ZKK_ID Signer's Name = example.com Signature = 0xfae5...(ω1) Authority Section : Signature = 0xd4cf...(ω2) Additional Section : Signature = 0xb2ac...(ω3) OPT </pre>

- 3) For every RRSIG_i in response R , do:
 - a) Check $\text{RRSIG}_i \rightarrow \text{RDATA} \rightarrow \text{Algorithm}$
 - If a signature algorithm is detected, execute the usual signature validation flow. If a KEM algorithm is found, proceed as below.
 - b) Let $\text{msg} := \text{RRSIG}_i \rightarrow \text{RDATA} \parallel \text{RR}_i(1) \parallel \dots$
 - where RDATA excludes Signature and $\text{RR}_i(j)$ is the j -th record in RRset_i
 - c) Compute $\omega' \leftarrow \text{MAC}_k(\text{msg})$
 - d) Verify $\omega' \stackrel{?}{=} \text{RRSIG}_i \rightarrow \text{RDATA} \rightarrow \text{Signature}$
- 4) If all RRSIG s are verified, mark R as secure.

E. Backward Compatibility

In this subsection, we examine what happens when only one of the end points implements the SL-DNSSEC protocol while the other one does not.

- *Protocol-aware Requester — Protocol-oblivious Responder:* The requester will not find a KEM ZKK_{pk} in the QTYPE DNSKEY response from the responder. It will then send a typical DNS query.
- *Protocol-oblivious Requester — Protocol-aware Responder:* A KEM ZKK_{pk} (along with KSK_{pk} and ZSK_{pk}) would be sent to the requester in the QTYPE DNSKEY response. However, ZKK_{pk} would be ignored as a key with an *unsupported* algorithm. The requester will then dispatch a usual DNS query. The responder, on not finding a KEM ciphertext in the query, will then proceed with the regular DNSSEC flow. In due time, when the requester receives a DNS response containing signatures, it will pick the relevant key (*i.e.* ZSK_{pk}) to perform the validation.

IV. SECURITY

We assess SL-DNSSEC’s security under the standard attacker model as used in a previous DNSSEC study [12]. In particular, the adversary’s ultimate aim is to induce the resolver to accept a malicious answer in Phase 4 (§III-D) of the protocol. All the capabilities of the (on-path) adversary, or lack thereof, are as listed below:

- It may eavesdrop on any exchanged packet.
- It may intercept, manipulate and re-send any exchanged packet as follows:
 - It may modify any HEADER bits.
 - It may modify the Question section.
 - It may remove/add/modify any resource record, including RRSIGs, DNSKEYs, or Type A or NS records.
- It cannot access any secret cryptographic keys.
- It can only do polynomial order computations.

Since the deployment of SL-DNSSEC does not depend on the zone’s parent, we omit the root (.) and the `com` TLD from the analysis. We also assume that a secure chain of trust exists from the root to `example.com` before SL-DNSSEC is deployed. Concretely, this secure chain of trust exists when:

- 1) (.)’s KSK_{pk} is the trust anchor on the resolver.
- 2) (.)’s KSK_{sk} signs (.)’s ZSK_{pk}
- 3) (.)’s ZSK_{sk} signs `com`’s DS containing a hash of `com`’s KSK_{pk}
- 4) `com`’s KSK_{sk} signs `com`’s ZSK_{pk}
- 5) `com`’s ZSK_{sk} signs `example.com`’s DS containing a hash of `example.com`’s KSK_{pk}

We now begin to scrutinize the SL-DNSSEC protocol between the resolver and the `example.com` ANS under the attacker model explicated earlier. Note that we only analyse attack surfaces that are unique to SL-DNSSEC. Attacks also applicable to regular DNSSEC, such as modifying HEADER or unsigned glue records, have already been appraised in [12].

A. Attacker alters DNSKEYs sent by ANS

During phase 1 (§III-A) of the protocol, an adversary runs `KEM.keygen()` to generate its own Zone KEM Key: $(ZKK_{pk}^{adv}, ZKK_{sk}^{adv})$ pair. On intercepting a QTYPE DNSKEY response sent by ANS to the resolver, the adversary may do either of the following changes to the DNSKEY RRset:

- Insert ZKK_{pk}^{adv} into the RRset.
- Replace the authentic ZKK_{pk} with ZKK_{pk}^{adv} .
- Delete ZKK_{pk} from the RRset.

However, assuming an EUF-CMA-secure signature scheme was used to sign the public key RRset, RRSIG validation will ultimately fail at the resolver’s end. Informally, EUF-CMA (Existential Unforgeability under Chosen Message Attack) security specifies that a polynomial time adversary cannot forge a signature on a new message, even after asking for signatures on arbitrary messages of its choice.

Alternatively, the adversary generates its own Key Signing Key $(KSK_{pk}^{adv}, KSK_{sk}^{adv})$ pair, and substitutes KSK_{pk} with KSK_{pk}^{adv} . Thereafter, it performs any of the three aforesaid amendments, and re-signs the modified RRset with KSK_{sk}^{adv} .

This time, the resolver will successfully verify the malicious DNSKEY RRset with KSK_{pk}^{adv} . However, assuming a *collision resistant* hash was used to compute the DS record of KSK_{pk} , the resolver will not be able to connect the trust of KSK_{pk}^{adv} with the parent, thus failing to complete the full validation. Note that the collision resistance of a hash function measures the ability of a polynomial adversary to find two distinct messages (here, KSKs) that hash to the same value.

B. Attacker alters SL-DNSSEC query sent by resolver

The usage of an IND-CCA-secure KEM (refer §II-C for the security definition) restricts the adversary in phase 2 (§III-B) to either of the manipulations underneath:

- Corrupt the ciphertext `ct` to `ct'`.
- Perform a KEM encapsulation against ZKK_{pk} of ANS to probabilistically obtain its own shared secret (ss_{adv}) and ciphertext (`ctadv`).

$$(ss_{adv}, ct_{adv}) \leftarrow \text{KEM.encap}(ZKK_{pk})$$

Then substitute `ct` with `ctadv` in the query.

- Remove DNSKEY record holding `ct` from the query.

In the first case, the failure behaviour of `KEM.decaps(ZKKsk, ct')` depends on the underlying KEM. In case of Kyber (refer [19], §4), if the re-encryption fails, the decapsulation will return a pseudo-random key $ss' = \text{hash}(z, ct')$, where z is a random secret seed. On the other hand, if `ct'` is a valid ciphertext, the decapsulation function will return a corresponding ss' . In either case, the ANS will derive an incorrect MAC key from ss' , eventually causing MAC failure on the resolver in phase 4 (§III-D).

Concerning the second scenario, the probability that the adversary obtains a ss_{adv} such that $ss_{adv} = ss$ is negligible. Therefore, with $ss_{adv} \neq ss$, the outcome will be the same as in the first case (*i.e.* MAC verification failure).

Finally, the last attempt will convert the query to a regular one. The ANS will deem the resolver to be SL-DNSSEC-oblivious, and thus revert to a signature-based flow.

C. Attacker alters SL-DNSSEC response sent by ANS

In phase 3 (§III-C), an adversary may tamper with the DNS response in the following manner:

- Modify the resource records in any of the three sections. For example, in case of a QTYPE A response, an adversary may change the IPv4 addresses present in Type A records.

Presuming an EUF-CMA-secure MAC was used to compute authentication tags on RRsets, the resolver will fail to validate the covering RRSIGs (containing MAC tags). Similar to signature schemes, EUF-CMA security for a MAC implies that even a polynomial attacker, which can query tags on chosen messages, cannot create a valid tag for a new message.

V. EVALUATION

A. Implementation

To assess the performance of SL-DNSSEC, we develop a daemon that runs on top of a DNS software (such as BIND or PowerDNS). Additionally, the daemon is designed to be *agnostic* to the said software (*i.e.* the underlying DNS provider can be swapped with a different one). With the daemon in place, no modifications are required to the DNS software stack, except for a small patch on the resolver’s side to detect whether the `Z` bit in the `HEADER` is on/off. The `Z` bit is what signals to the DNS software that the response has been successfully SL-DNSSEC-validated by the daemon. We now succinctly discuss the functionality of the daemon in question.

Daemon. Figure 4 illustrates a SL-DNSSEC validation being performed with the aid of the daemon. In all our experiments, we pre-generate and hardcode the KEM keys in the daemon. In actual practice, the KEM public key and the corresponding signature would be fetched by the resolver via a `QTYPE DNSKEY` query as discussed in §III-A.

The daemon performs all SL-DNSSEC related operations independently of the DNS software (here, BIND). Observe that the daemon on the ANS sets `OPT → CLASS` to 65507 (the maximum UDP payload size over IPv4) before forwarding the query to BIND. This is to allow the retrieval of the full DNS response from BIND⁵ without truncation. If the size of BIND’s response exceeds the resolver’s (originally) advertised EDNS0 buffer size (here, 1232), the daemon replaces the signatures with MACs as outlined in §III-C.

On the resolver, the daemon performs the SL-DNSSEC validation of DNS records (as elucidated in §III-D) and sets `HEADER → Z = 1` in case of a successful outcome.

Software Setup. We use the source code of QBF [56] as base to build the SL-DNSSEC daemon. The DNS software is a BIND 9.19.17 fork [7] which supports NIST level I PQC signatures. In the fork, we further add support for:

- 1) Level V Falcon and Dilithium schemes
- 2) Detecting the `Z` bit in the `HEADER`

The cryptographic stack is `openssl 3.2`, `liboqs 0.10.0` [66] and `oqs-provider 0.6.0`. The daemon is written in C and uses `libnetfilter-queue` to intercept incoming and outgoing DNS packets. Docker 4.29 is used for constructing the network scenario (described below). To simulate network bandwidth and latency, we use Linux’s `tc` utility. DNS queries are issued using `dig`. Communication statistics are obtained with `ip` command. All experiments are run on a MacBook Air M1 laptop with 8 GB of RAM.

Network Scenario. The DNS network contains the following four participants: 1) A client 2) A resolver 3) A root (`.`) nameserver 4) An `example` authoritative nameserver (ANS). We skip configuring a `com` TLD to reduce complexity. Each participant runs as a private Ubuntu 22.04 Docker container with experiment-specific bandwidth and latency constraints. Additionally, the SL-DNSSEC daemon is installed on both the resolver and the ANS containers.

The EDNS0 buffer size is set to the recommended value of 1232 bytes. For simplicity, each zone is signed with a single algorithm and has one ZSK and one KSK. In addition, the daemons on both the resolver and the ANS are pre-configured with the requisite KEM ZKK keys.

The zone file served by the ANS contains 10 Type A records, each with a unique domain name and an associated RRSIG. In its `named.conf`, the ANS is configured with `minimal-responses no-auth-recursive`; (the default setting that ships with BIND) which means that it will be *as complete as possible* while generating responses for iterative queries. Such a response is called *non-minimal* and represents the worst-case scenario in terms of message size. Refer §III-C for the number and the type of records contained in a *non-minimal* `QTYPE A` response returned by the ANS in the described setup. To facilitate modifications to DNS messages without readjusting compression name pointers, we also set `message-compression no`; in `named.conf`.

B. Experiments and Results

We now comprehensively assess SL-DNSSEC’s performance against signature-based DNSSEC in terms of bandwidth usage and resolution times. We conduct two experiments targeting NIST security level I and V, respectively. Before the start of an experiment, the resolver pre-fetches the Type `DNSKEY` and `NS` of all the zones, including the `DS` record of `example` zone. The implication is that the resolver directly contacts the ANS in order to resolve the client’s query, rather than starting the lookup process all the way up from the root.

Signature-based DNSSEC instances are run over two transport mechanisms:

- 1) Standard DNS (SD) over UDP with a fallback to TCP in case of a truncated (`TC`) response
- 2) An upper-layer UDP-only fragmentation scheme such as ARRF/QBF

We exclude ARRF from our experiments as its performance can be extrapolated from that of QBF. Both schemes (since they only differ in the way fragments are packaged) have roughly the same bandwidth usage, with QBF being a round trip faster than ARRF. The number of packets sent and received also remains within ± 1 margin, respectively.

Each experiment consists of two main stages: 1) Measure the bandwidth consumption during a single query resolution, and 2) Measure the mean resolution time of 10 queries.

Measuring bandwidth usage. We send a `QTYPE A` DNS query from the client to the resolver. At the resolver’s Ethernet interface, we then assess the network communication with the ANS in terms of:

- Number of packets (technically, frames) in transmit (TX) and receive (RX)
- Number of bytes in transmit (TX) and receive (RX). Note that these values include:
 - 1) 14-byte Ethernet header
 - 2) 20-byte IPv4 header
 - 3) 8-byte UDP header or 32-byte TCP header (40-byte in case of SYN and SYN-ACK)

⁵Increases in BIND buffer sizes were also required.

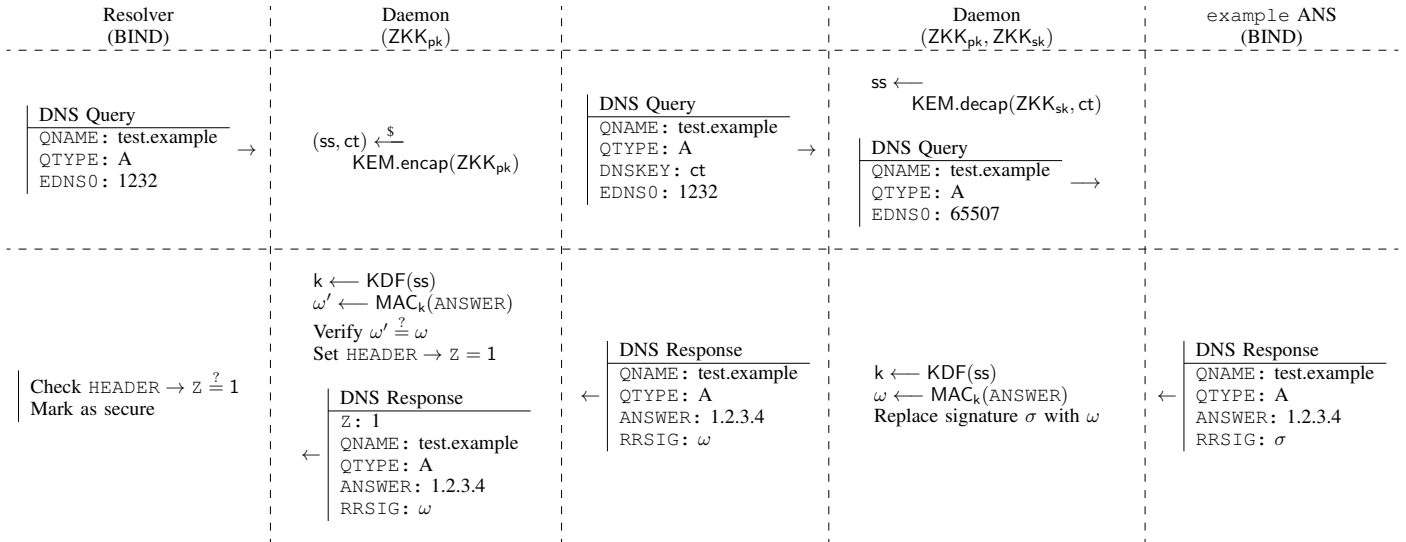


Fig. 4: An overview of SL-DNSSEC validation via Daemon (NIST Level I)

- Transport protocol used. Here, TCP* indicates a TCP fallback wherein the first round trip is over UDP while the subsequent ones are over TCP.

Measuring resolution time. We measure the query resolution speed under two network conditions⁶:

- 1) High Bandwidth (100 Mbps), Low Latency (10 ms)
- 2) Low Bandwidth (1 Mbps), High Latency (100 ms)

Specifically, we issue 10 QTYPE A DNS queries from the client to the resolver and calculate the mean resolution time. That is, the average time elapsed between the client sending its query and subsequently receiving a DNSSEC validated response (with HEADER → AD set) from the resolver.

1) *Experiment 1:* We target NIST level I parameters. To instantiate SL-DNSSEC, we use the following primitives:

- **Post-Quantum KEM:** Kyber-512
- **KDF:** HKDF-SHA-256 [43]
- **MAC:** HMAC-SHA-256 [42]

To determine how SL-DNSSEC fares against signature-based DNSSEC, we sign the zone file with the schemes below:

- Pre-Quantum: RSA-2048, ECDSA-P256
- Post-Quantum: Falcon-512, Dilithium-2, SPHINCS⁺-128s

Results and Discussion. All the results of the experiment are catalogued in Tables (XI, XII) and Figure 5. We observe that the Kyber-HMAC instantiation of SL-DNSSEC, while additionally providing a level I post-quantum security, is virtually equivalent to RSA-2048 in terms of total bytes exchanged and resolution times. In fact, out of all the tested mechanisms, Kyber-HMAC has the smallest response size (RX), even beating out ECDSA-SD.

TABLE XI: A comparison of bandwidth usage. SD : Standard DNS. TCP* : TCP fallback. (NIST Level I)

Method	Via	Pkts. Sent TX	Pkts. Rcvd. RX	Bytes Sent TX	Bytes Rcvd. RX	BAF	Bytes Xferred TX+RX
ECDSA-SD	UDP	1	1	84	512	6.1	596
RSA-SD	UDP	1	1	84	1088	13	1172
Falcon-SD	TCP*	8	6	598	2700	-	3298
Dilithium-SD	TCP*	13	11	928	8292	-	9220
SPHINCS ⁺ -SD	TCP*	24	24	1654	25389	-	27043
Falcon-QBF	UDP	3	3	258	2788	10.8	3046
Dilithium-QBF	UDP	8	8	693	9225	13.3	9918
SPHINCS ⁺ -QBF	UDP	23	23	2012	28321	14.1	30333
Kyber-HMAC	UDP	1	1	867	416	0.48	1283

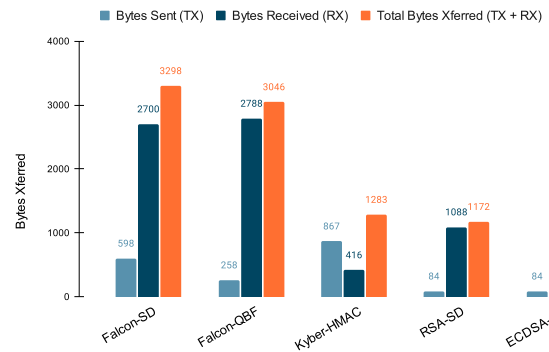


Fig. 5: A visualization of Table XI

⁶Network RTT = 2 × Latency.

TABLE XII: A comparison of query resolution times. SD denotes Standard DNS. (NIST Level I)

Method	100 Mbps, 10 ms Avg. Resolution Time (± 1 ms)	1 Mbps, 100 ms Avg. Resolution Time (± 2 ms)
ECDSA-SD	44	407
RSA-SD	44	408
Falcon-SD	89	811
Dilithium-SD	89	817
SPHINCS ⁺ -SD	111	1025
Falcon-QBF	45	410
Dilithium-QBF	46	415
SPHINCS ⁺ -QBF	48	436
Kyber-HMAC	44	408

Concerning NIST recommended signatures, Kyber-HMAC requires less than half the bandwidth of Falcon instances. On bringing Dilithium and SPHINCS⁺ into the picture, the bandwidth savings become even more dramatic (i.e. 86% and 95%, respectively). The upshot is that, with SL-DNSSEC, servers will not need to upgrade to a higher bandwidth connection, thus shrinking the operational costs.

Furthermore, Kyber-HMAC (SL-DNSSEC) remains immune against being exploited as a DNS amplifier. Observe that Kyber-HMAC transmits a large query owing to the KEM ciphertext contained therein. Due to a response being *smaller* than its query, the Bandwidth Amplification Factor (BAF) becomes < 1 , resulting in a negative return on bandwidth investment for a prospective attacker. On the other hand, all signature-based methods over UDP can be potentially exploited by an attacker for DDoS amplifications attacks.

With reference to post-quantum signatures over Standard DNS (SD), we observe a slowdown of at least 50% compared to other setups. This is because the DNS response containing PQC signatures always exceeds the EDNS buffer size of 1232 bytes. Consequently, the initial UDP trip is wasted (due to the response being marked truncated) and overall resolution times further increased (due to the ensuing 3-way TCP handshake).

Interestingly, SPHINCS⁺-SD even incurs an extra round trip compared to Falcon-SD and Dilithium-SD. This is because the size of a SPHINCS⁺ QTYPE A response exceeds the `initcwnd` (initial congestion window) of 10 segments set in the TCP slow start algorithm [17], [22]. Given the default MSS (Maximum Segment Size) of 1220 bytes, the size of `initcwnd` comes out to be $10 \times 1220 = 12.2$ KB. The repercussion of exceeding this `initcwnd` is that after sending about 12.2 KB of data, the nameserver waits for the resolver to ACK (acknowledge) the received packets before continuing with the rest of the transmission.

While QBF matches the resolution speeds of classical DNSSEC, it exchanges multiple DNS/UDP packets in proportion to the size of the original (un-fragmented) response. Given the absence of any flow and congestion control mechanisms in UDP, this torrent of packets can potentially exhaust the network resources of busy resolvers/nameservers and overwhelm middleboxes, whilst also increasing the chances of the session requiring more round-trips due to unanticipated packet drops.

TABLE XIII: A size comparison (in bytes) of signature (sig) / ciphertext (ct) and public key (pk) of various algorithms.

Algorithm	sig / ct	pk
Falcon-1024	1280	1793
Dilithium-5	4595	2592
Kyber-1024	1568	1568

Somewhat surprisingly, RSA-SD has a higher BAF than Falcon-QBF, despite Falcon’s signature being almost $2.6\times$ the size of RSA’s. This is because QBF sends multiple full-fledged DNS queries, which increases the amount of TX bytes (the denominator), thereby amortizing BAF.

Finally, observe that QBF starts to consume slightly more bandwidth than Standard DNS (SD) as the number of exchanged packets grow. This is because QBF first envelops a signature fragment in an RRSIG record. This RRSIG is then inserted in a conventional DNS message (along with its concomitant 12-byte HEADER, Question section, OPT record, etc.) resulting in a data overhead. TCP, on the other hand, is a continuous byte-stream of the original DNS response.

2) *Experiment 2*: To assess SL-DNSSEC’s scalability, we now target NIST level V. This is the highest security level on offer and is *likely excessive* for DNSSEC [13].

Choice of Primitives. The updated parameters are:

- **Post-Quantum KEM:** Kyber-1024
- **KDF:** HKDF-SHA-512 [43]
- **MAC:** HMAC-SHA-512 [42]

To compare SL-DNSSEC with signature-based DNSSEC, we sign the zone file with Falcon-1024 and Dilithium-5.

Table XIII compares the object sizes of various NIST level V signature and KEM schemes. We omit testing SPHINCS⁺ since the resulting response would exceed 64 KB, the maximum possible size for a DNS message. Furthermore, since a DNS query carrying a Kyber ciphertext of 1568 bytes would exceed the recommended threshold of 1232, we adapt QBF [56] to split ct into two DNS queries, as sketched in Fig. 6.

Results and Discussion. All the findings of Experiment 2 are rendered in Tables (XIV, XV) and Figure 7. Compared to Falcon and Dilithium instances, Kyber-HMAC (SL-DNSSEC) still manages to cut bandwidth consumption by about 56% and 86%, respectively. Furthermore, thanks to small MACs, the response size (RX) in Kyber-HMAC increases by only 96 bytes despite the big jump in security level. However, to meet the UDP size constraints of 1232 bytes, the resolver daemon unavoidably has to dispatch an extra query.

Lastly, while Falcon-SD remains consistent with its resolution speeds, Dilithium-SD suffers a penalty of an extra round-trip because of exceeding TCP’s `initcwnd`. Notice also that the BAF in Falcon-QBF and Dilithium-QBF is the same, despite a large discrepancy in the underlying signature sizes. Again, this is to be ascribed to the higher amount of TX bytes (due to more DNS queries) in the latter case.

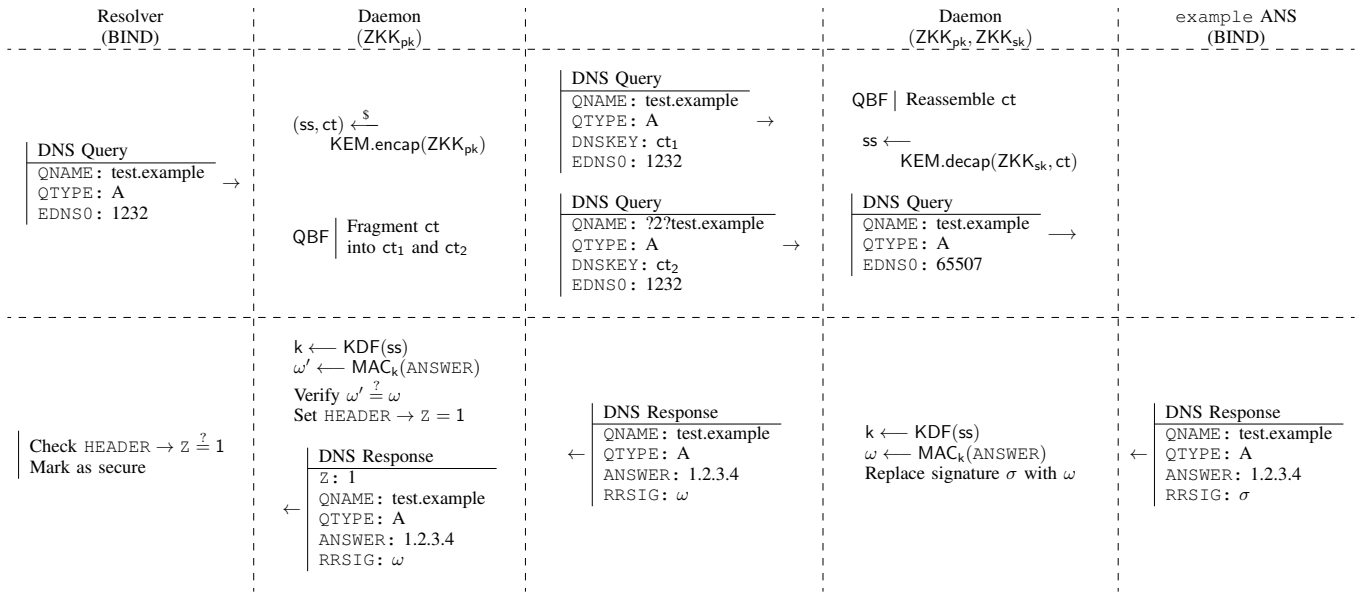


Fig. 6: An overview of SL-DNSSEC validation via Daemon (NIST Level V)

TABLE XIV: A comparison of bandwidth usage. SD : Standard DNS. TCP* : TCP fallback. (NIST Level V)

Method	Via	Pkts. Sent TX	Pkts. Rcvd. RX	Bytes Sent TX	Bytes Rcvd. RX	BAF	Bytes Xferred TX+RX
Falcon-SD	TCP*	10	8	730	4674	-	5404
Dilithium-SD	TCP*	18	16	1258	15147	-	16405
Falcon-QBF	UDP	4	4	345	4865	14.1	5210
Dilithium-QBF	UDP	14	14	1220	17165	14.1	18385
Kyber-HMAC	UDP	2	1	1766	512	0.29	2278

TABLE XV: A comparison of query resolution times. SD denotes Standard DNS. (NIST Level V)

Method	100 Mbps, 10 ms Avg. Resolution Time (± 1 ms)	1 Mbps, 100 ms Avg. Resolution Time (± 2 ms)
Falcon-SD	89	812
Dilithium-SD	111	1014
Falcon-QBF	45	411
Dilithium-QBF	47	426
Kyber-HMAC	44	409

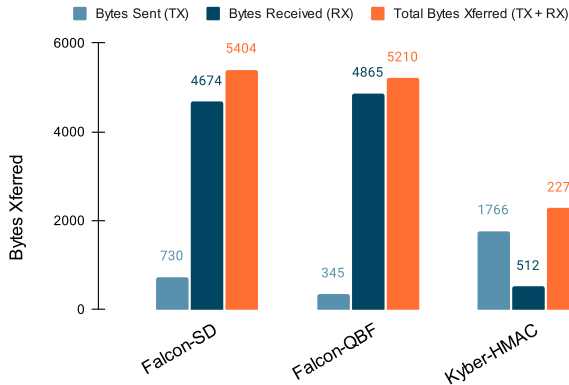


Fig. 7: A visualization of Table XIV

VI. OTHER CONSIDERATIONS

A. Fetching DNSKEYs from Nameservers

It is important to note that SL-DNSSEC relies upon a PQC signature on the DNSKEY RRset to ascertain the authenticity of the KEM public key (ZKK_{pk}) contained therein.

Although DNS responses in SL-DNSSEC remain well under 1232 bytes, the initial QTYPE DNSKEY response from a nameserver containing multiple DNSKEYs and one or more RRSIGs will likely not respect the aforesaid size ceiling.

Therefore, similar to regular DNSSEC, retrieving the DNSKEYs of a zone may entail the use of either one of the following methods:

- 1) An upper-layer fragmentation scheme (ARRF/QBF)
- 2) Standard DNS with a fallback to TCP
- 3) An out-of-band distribution via HTTP or FTP

Fortunately, this is not much of a concern since DNSKEYs are fetched infrequently owing to their higher caching TTLs.

B. Managing Keys on Nameservers

Considerations that are pertinent to DNSSEC in live (on-the-fly) signing mode also remain applicable to SL-DNSSEC. Specifically, since KEM keys will be stored on nameservers that connect to the Internet (which increases the overall attack surface), a hardware security module (HSM) is therefore recommended for securely managing the keys.

TABLE XVI: A comparison of cryptographic operations.

	SL-DNSSEC	DNSSEC Offline Signing	DNSSEC Live Signing
Resolver Query	KEM.encap()	-	-
Nameserver Response	KEM.decap() KDF() MAC()	-	SIG.sign()
Resolver Validation	KDF() MAC()	SIG.verify()	SIG.verify()

C. Computational Requirements

Table XVI contrasts the computations performed in SL-DNSSEC and conventional DNSSEC. Typically, KEM and MAC operations are more efficient than their signature counterparts (refer [70], pp. 197-198). However, concretely comparing the computational time of the two protocols is not straightforward, since implementations and hardware are constantly improving over time. We therefore leave this as an interesting avenue for further research.

VII. CONCLUSION

We presented the SL-DNSSEC protocol: a signature-free alternative for performing DNSSEC validations in a single query/response fashion. Leveraging a quantum-safe KEM and a MAC, SL-DNSSEC achieves NIST level I security while having analogous bandwidth usage and resolution speeds to that of RSA-2048. Moreover, owing to smaller responses than queries, SL-DNSSEC remains unusable as a DNS amplifier.

REFERENCES

- [1] “Dan kaminsky, black ops 2008: It’s the end of the cache as we know it,” <https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf>, accessed: 2024-07-09.
- [2] “The ddos that almost broke the internet,” <https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet>, accessed: 2024-07-09.
- [3] “Dns flag day 2020,” <https://www.dnsflagday.net/2020/>.
- [4] “The double ratchet algorithm,” <https://signal.org/docs/specifications/doublerratchet/>.
- [5] “Is large-scale dns over tcp practical?” <https://ripe76.ripe.net/presentations/95-jonglez-dns-tcp-ripe76.pdf>, accessed: 2024-07-09.
- [6] “Noise protocol framework,” <https://noiseprotocol.org/noise.html>.
- [7] “Oqs-bind,” <https://github.com/Martyrshot/OQS-bind>.
- [8] “Defragmenting dns - determining the optimal maximum udp response size for dns,” 2020, accessed: 2024-07-09. [Online]. Available: <https://indico.dns-oarc.net/event/36/contributions/776/>
- [9] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. Schoen, and B. Warren, “Let’s encrypt: An automated certificate authority to encrypt the entire web,” in *SIGSAC CCS*, 2019.
- [10] S. Ariyapperuma and C. J. Mitchell, “Security vulnerabilities in dns and dnssec,” in *ARES*, 2007.
- [11] D. Atkins and R. Austein, “Threat analysis of the domain name system (dns),” RFC 3833, 2004.
- [12] J. Bau and J. C. Mitchell, “A security evaluation of DNSSEC with NSEC3,” in *NDSS*, 2010.
- [13] G. Beernink, “Taking the quantum leap: Preparing dnssec for post quantum cryptography,” Master’s thesis, University of Twente, 2022. [Online]. Available: <http://essay.utwente.nl/89509/>
- [14] M. Bellare, R. Canetti, and H. Krawczyk, “A modular approach to the design and analysis of authentication and key exchange protocols,” in *STOC*, 1998.
- [15] M. Bellare and P. Rogaway, “Entity authentication and key distribution,” in *CRYPTO*, 1994.
- [16] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, “The sphincs+ signature framework,” in *SIGSAC CCS*, 2019.
- [17] E. Blanton, D. V. Paxson, and M. Allman, “Tcp congestion control,” RFC 5681, 2009.
- [18] R. Bonica, F. Baker, G. Huston, B. Hinden, O. Trøan, and F. Gont, “Ip fragmentation considered fragile,” RFC 8900, 2020.
- [19] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, “Crystals - kyber: A cca-secure module-lattice-based kem,” in *EuroS&P*, 2018.
- [20] R. Bush and R. Austein, “The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1,” RFC 8210, 2017.
- [21] R. Canetti and H. Krawczyk, “Analysis of key-exchange protocols and their use for building secure channels,” in *EUROCRYPT*, 2001.
- [22] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, “Increasing tcp’s initial window,” RFC 6928, 2013.
- [23] T. Chung, R. Van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, “A longitudinal, end-to-end view of the dnssec ecosystem,” in *USENIX*, 2017.
- [24] J. da Silva Damas, M. Graff, and P. A. Vixie, “Extension mechanisms for dns (edns(0)),” RFC 6891, 2013.
- [25] C. de Saint Guilhem, N. P. Smart, and B. Warinschi, “Generic forward-secure key agreement without signatures,” in *ISC*, 2017.
- [26] P. Dikshit, M. Kosek, N. Faulhaber, J. Sengupta, and V. Bajpai, “Evaluating dns resiliency and responsiveness with truncation, fragmentation & dotcp fallback,” *IEEE TNSM*, 2024.
- [27] J. A. Donenfeld, “Wireguard: Next generation kernel network tunnel,” in *NDSS*, 2017.
- [28] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals-dilithium: A lattice-based digital signature scheme,” *IACR TCHES*, 2018.
- [29] D. E. Eastlake and M. P. Andrews, “Domain name system (dns) cookies,” RFC 7873, 2016.
- [30] A. Fregly, J. Harvey, B. S. Kaliski Jr., and S. Sheth, “Merkle tree ladder mode: Reducing the size impact of nist pqc signature algorithms in practice,” in *CT-RSA*, 2023.
- [31] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama, “Strongly secure authenticated key exchange from factoring, codes, and lattices,” in *PKC*, 2012.
- [32] J. Goertzen and D. Stebila, “Post-quantum signatures in DNSSEC via request-based fragmentation,” in *PQCrypto*, 2023.
- [33] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *STOC*, 1996.
- [34] A. Herzberg and H. Shulman, “Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org,” in *IEEE CNS*, 2013.
- [35] P. E. Hoffman and P. McManus, “Dns queries over https (doh),” RFC 8484, 2018.
- [36] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman, “Specification for dns over transport layer security (tls),” RFC 7858, 2016.
- [37] C. Huitema, S. Dickinson, and A. Mankin, “Dns over dedicated quic connections,” RFC 9250, 2022.
- [38] P. Jeitner and H. Shulman, “Injection attacks reloaded: Tunnelling malicious payloads over DNS,” in *USENIX*, 2021.
- [39] P. Kampanakis and T. Lepoint, “Vision paper: Do we need to change some things?” in *SSR*, 2023.
- [40] C. Kaufman, R. Perlman, and B. Sommerfeld, “Dos protection for udp-based protocols,” in *SIGSAC CCS*, 2003.
- [41] M. Kosek, T. V. Doan, S. Huber, and V. Bajpai, “Measuring dns over tcp in the era of increasing dns response sizes: a view from the edge,” *SIGCOMM CCR*, 2022.
- [42] D. H. Krawczyk, M. Bellare, and R. Canetti, “Hmac: Keyed-hashing for message authentication,” RFC 2104, 1997.
- [43] D. H. Krawczyk and P. Eronen, “Hmac-based extract-and-expand key derivation function (hkdf),” RFC 5869, 2010.

- [44] H. Krawczyk, “Skeme: a versatile secure key exchange mechanism for internet,” in *NDSS*, 1996.
- [45] —, “Hmqv: A high-performance secure diffie-hellman protocol,” in *CRYPTO*, 2005.
- [46] H. Krawczyk and H. Wee, “The optls protocol and tls 1.3,” in *EuroS&P*, 2016.
- [47] B. LaMacchia, K. Lauter, and A. Mityagin, “Stronger security of authenticated key exchange,” in *ProvSec*, 2007.
- [48] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, “An efficient protocol for authenticated key agreement,” *DCC*, 2003.
- [49] M. Lepinski and S. Kent, “An Infrastructure to Support Secure Internet Routing,” RFC 6480, 2012.
- [50] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, “Dns cache poisoning attack reloaded: Revolutions with side channels,” in *SIGSAC CCS*, 2020.
- [51] K. Man, X. Zhou, and Z. Qian, “Dns cache poisoning attack: Resurrections with side channels,” in *SIGSAC CCS*, 2021.
- [52] J. Mao, M. Rabinovich, and K. Schomp, “Assessing support for dns-over-tcp in the wild,” in *PAM*, 2022.
- [53] G. C. M. Moura, M. Müller, M. Davids, M. Wullink, and C. Hesselman, “Fragmentation, truncation, and timeouts: Are large dns messages falling to bits?” in *PAM*, 2021.
- [54] M. Müller, J. de Jong, M. van Heesch, B. Overeinder, and R. van Rijswijk-Deij, “Retrofitting post-quantum cryptography in internet protocols: a case study of dnssec,” *SIGCOMM CCR*, 2020.
- [55] T. Prest, P. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, “Falcon. tech. rep., national institute of standards and technology, available at,” <https://src.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>, 2022.
- [56] A. S. Rawat and M. P. Jhanwar, “Post-quantum dnssec over udp via qname-based fragmentation,” in *SPACE*, 2023.
- [57] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, “Dns security introduction and requirements,” RFC 4033, 2005.
- [58] —, “Protocol modifications for the dns security extensions,” RFC 4035, 2005.
- [59] —, “Resource records for the dns security extensions,” RFC 4034, 2005.
- [60] C. Rossow, “Amplification hell: Revisiting network protocols for ddos abuse,” in *NDSS*, 2014.
- [61] P. Schwabe, D. Stebila, and T. Wiggers, “Post-quantum tls without handshake signatures,” in *SIGSAC CCS*, 2020.
- [62] —, “More efficient post-quantum kemtls with pre-distributed public keys,” in *ESORICS*, 2021.
- [63] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SICOMP*, 1997.
- [64] M. Sivaraman, S. Kerr, and L. Song, “Dns message fragments,” <https://datatracker.ietf.org/doc/draft-muks-dns-message-fragments/00/>.
- [65] L. Song and S. Wang, “Atr: Additional truncation response for large dns response,” <https://datatracker.ietf.org/doc/draft-song-atr-large-resp/03/>.
- [66] D. Stebila and M. Mosca, “Post-quantum key exchange for the internet and the open quantum safe project,” in *SAC*, 2017.
- [67] G. Van Den Broek, R. Van Rijswijk-Deij, A. Sperotto, and A. Pras, “Dnssec meets real world: dealing with unreachability caused by fragmentation,” *IEEE Communications Magazine*, 2014.
- [68] R. van Rijswijk-Deij, M. Jonker, A. Sperotto, and A. Pras, “A high-performance, scalable infrastructure for large-scale active dns measurements,” *IEEE JSAC*, 2016.
- [69] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, “Dnssec and its potential for ddos attacks: a comprehensive measurement study,” in *IMC*, 2014.
- [70] T. Wiggers, “Post-quantum tls,” Ph.D. dissertation, Radboud University, 2024. [Online]. Available: <https://thomwiggers.nl/publication/thesis/>