Post-Quantum DNSSEC over UDP via QNAME-Based Fragmentation

Aditya Singh Rawat
Mahabir Prasad Jhanwar
aditya.rawat_phd21@ashoka.edu.in
mahavir.jhawar@ashoka.edu.in
Ashoka University
Sonipat, India

Abstract

In a typical network, a DNS(SEC) message over 1232 bytes would either be fragmented into several UDP/IP packets or require a retransmit over TCP. Unfortunately, IP fragmentation is considered unreliable and a non-trivial number of servers do not support TCP.

We present `QNAME`-Based Fragmentation (QBF): a DNS layer fragmentation scheme that fragments/re-assembles large post-quantum DNS(SEC) messages over UDP in just 1 round-trip while using only standard DNS records. Our experiments show that DNSSEC over QBF, with either Falcon-512, Dilithium-2 or SPHINCS$^+$ as the zone signing algorithm, is practically as fast as the currently deployed ECDSA-P256 and RSA-2048 setups in resolving `QTYPE A` queries.

1 Introduction

A series of developments in quantum computing has prompted the need to replace the cryptographic algorithms based on the believed hardness of integer factorization and discrete logarithm problem (DLP). Currently, many Internet protocols rely on these algorithms to 1) ensure message confidentiality and integrity, and 2) to authenticate the communicating parties.

DNS Security Extensions (DNSSEC) [21–23], being one such protocol, provides authenticity and integrity for messages exchanged in the Domain Name System (DNS). In its main capacity, DNS translates a human-readable domain name (`www.example.com`) to a machine-understandable IP addresses (`1.2.3.4`).

Without DNSSEC, the recipients of a DNS response cannot verify the integrity of the IP address contained therein, and thus risk being misdirected to a malicious website [1, 4]. Unfortunately DNSSEC, owing to its use of classical algorithms, can be rendered completely ineffective by a sufficiently capable quantum computer.

Although DNS over TLS [14], DNS over HTTPS [13], and DNS over QUIC [15] have been proposed, they are not a substitute for DNSSEC. The former establish an encrypted and authenticated channel between a client and a resolver. On the other hand, DNSSEC 1) works between resolvers and nameservers, and 2) guarantees the veracity of DNS records by forming a chain of trust up to the root.

In its efforts to sustain digital security in the face of quantum computers, the National Institute of Standards and Technology (NIST) has selected Crystals-Kyber [7] as Key Encapsulation Mechanism (KEM) and Crystals-Dilithium [10], Falcon [20] and SPHINCS$^+$ [5] as signature algorithms. In comparison to their classical siblings however, these algorithms (colloquially referred to as PQC (*i.e.* Post-Quantum Cryptography), have strikingly larger public key and signature sizes as elucidated in Table 1.

Table 1: A comparison of signature (sig) and public key (pk) sizes (in bytes) of various signature schemes.

Algorithm	Assump.	Quantum-safe	pk	sig
ECDSA-P256	ECDLP	✗	64	64
RSA-2048	Factoring	✗	260	256
Falcon-512	Lattice	✓	897	690
Dilithium-2	Lattice	✓	1312	2420
SPHINCS$^+$-128s	Hash	✓	32	7856

1.1 DNS Size Constraints

This increase in sizes of signatures and public keys (and consequently of DNS messages) bears major implications for DNSSEC [16]. A DNS over UDP message, as originally standardized, was restricted to a maximum size of 512 bytes. Bearing in mind the headroom required by DNSSEC (for conveying signatures and public keys), this size ceiling was later increased to a *theoretical* value of 64 KB with Extension Mechanisms for DNS (EDNS0) [8].

However, a DNS over UDP message exceeding 1232 bytes in size usually triggers IP fragmentation on most network links [2, 19]. This upper bound of 1232 has been derived as follows: 1280 (IPv6 minimum MTU) − 40 (IPv6 Header) − 8 (UDP Header).

It is evident that DNSSEC messages carrying post-quantum data will easily exceed 1232 bytes in size, and thus be fragmented by the network. Unfortunately, IP fragmentation is considered to be both unreliable (fragments may never arrive) and insecure (fragments can be spoofed) [6]. Moreover, Broek *et al.* [27] have noted that up to 10% of the resolvers fail to handle fragments correctly.

The other alternative for sending large DNS messages without resorting to IP fragmentation is via TCP. In a usual DNS flow, when a response size exceeds the resolver's advertised EDNS0 buffer size (*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 over TCP.

Unfortunately, up to 11% of nameservers have been found to lack TCP support by [28]. The report of [19] additionally remarks that TCP/53 connections could even be blocked by intruding middleboxes. In the surveys of [9, 18], 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 [3, 17], thus putting a limit on the number of TCP connections a DNS server might be able to handle concurrently.

Related Work

Many proposals have been put forward that address the two foregoing issues of network layer fragmentation and TCP non-availability, respectively, by fragmenting at the application (DNS) layer. The implication of this approach is that the nameserver becomes responsible for the fragmentation of a DNS response and the resolver for the subsequent reassembly thereof.

Sivaraman *et al.* [24] fragmented a large DNS response across multiple UDP datagrams and transmitted each fragment sequentially. On the other hand, though not strictly a fragmentation scheme, ATR [25] sent an additional TC response to trigger a TCP fallback on the client, in case the original response failed to arrive.

Unfortunately, both of these schemes failed to get standardized because they sent multiple responses to a single request. Many firewalls are configured to accept only one response packet per query. Moreover, many resolvers close their sockets immediately after receiving the *first* response packet. In such a case, there was also a risk of ICMP flooding since the dropped packets will generate multiple *destination unreachable* messages.

A recent work, called ARRF [11], fragmented DNS resource records and addressed the principal shortcoming of earlier approaches by sending additional messages only upon *request*. Since each extra response has its own query, concerns about intruding firewalls and ICMP flooding are mitigated.

Unfortunately, ARRF suffers from various limitations, as acknowledged by its authors in [11]. Firstly, ARRF introduces a new type of DNS resource record called Resource Record Fragment (RRFRAG), which being non-standard, has the potential to be rejected by certain middleboxes. Secondly, ARRF is vulnerable to memory exhaustion attacks. Lastly, ARRF needs a minimum of two round-trips to reconstruct the full DNS message.

1.2 Our Contributions

In this work, we propose a fragmentation scheme called QNAME-Based Fragmentation (QBF). Similar to previous approaches, QBF also performs fragmentation at the application (DNS) layer, thus obviating concerns of IP fragmentation and lack of TCP support. Moreover, QBF is *request*-based like ARRF [11] in that each extra response has its own related query. Hence, messages sent by QBF are not prone to the issues of firewall filtering or ICMP flooding.

In contrast to ARRF however, QBF offers the following benefits:

- **Backward Compatibility**: Unlike ARRF which uses a non-standard resource record called RRFRAG, QBF uses only standard DNS records. Thus, QBF messages are not susceptible to getting blocked by strict middleboxes which inspect the resource records of a DNS message.
- **Security against Memory Exhaustion Attacks**: Owing to its design, ARRF exposes an attack surface wherein an adversary can deplete resolver's memory by injecting malicious RRFRAGs in the initial response. However, QBF is not vulnerable to such attacks.
- **1-RTT Resolution**: DNSSEC over QBF is $\sim 2\times$ as fast as Standard DNS (SD) in resolving QTYPE A queries. Moreover QBF, being 30% faster than ARRF, matches the resolution speeds of the presently deployed RSA-2048. See Fig. 1.

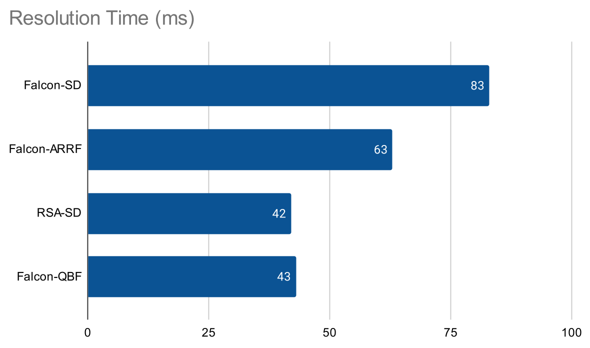

Figure 1: A comparison of DNSSEC resolution times

Our implementation of QBF is a daemon that runs on top of the DNS software (such as BIND9 or PowerDNS) of nameservers and resolvers. It fragments/reassembles large DNS messages (as and when needed) and requires *no* modifications to the underlying DNS stack or zone files. The source code germane to this work is available at: https://github.com/aditya-asr/qbf_src.

2 Preliminaries

Notations. The term *resource record* (RR) is often referred to as simply a *record.* || represents concatenation. X $\longrightarrow$ Y denotes member Y of an abstract structure X. 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).

2.1 Domain Name System (DNS)

Consider a canonical domain name: `www.example.com.` (with the trailing dot). Each label: (`www`), (`example`), (`com`) and (`.`) 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 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 (`.`) ANS, which subsequently responds with the following *glue* (referral) records: 1) A Type NS record containing the domain name of `com` ANS 2) A Type A record containing the IP of `com` ANS.
(2) It sends a QTYPE NS query to the `com` ANS, which then responds with the following *glue* records: 1) A Type NS record containing the domain name of `example.com` ANS 2) A Type A record containing the IP of `example.com` ANS.
(3) It sends a QTYPE A query to the `example.com` ANS, 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.

Table 2: DNS HEADER **Wire Format**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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)
— `AD`: whether response has authenticated data (1) or not (0)
— `RCode`: (0) - no error; (1) or `FORMERR` - query was malformed; (3) or `NXDOMAIN` - domain name does not exist

Wire Format. A DNS message is divided into five sections: HEADER, Question, Answer, Authority, and Additional. HEADER is always present and has a constant size of 12 bytes. Table 2 presents the wire format of a DNS HEADER. The Question section consists of the following fields: `QNAME` (specifies the domain name encoded in the DNS name notation. For example, `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` for 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), `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.

DNS Message Size. DNS messages are sent over the Internet using a series of layers. Initially, they are placed into UDP datagrams, which in turn are placed inside IP packets. These IP packets become the payload of frames at the link layer. However, there is a limit to the payload size of these frames based on the Maximum Transmission Unit (MTU) of the link they are travelling over. If a frame's payload is too large for the link's MTU, a router must break it into smaller IP packets, resulting in fragmentation. These fragmented IP packets travel independently to their destination. Since DNS relies on UDP, which does not guarantee a reliable communication like TCP, any loss of fragmented IP packets can cause transmission failures. Even when fragmentation does work, it may not be secure. It is theoretically possible to spoof parts of a fragmented DNS message, without an easy detection by the receiver.

To address these issues, there are two solutions: a) configure servers to limit the size of DNS messages sent over UDP to ensure that they do not trigger fragmentation on typical network links; b) ensure that DNS servers can switch from UDP to TCP when a DNS response is too large to fit within the limited buffer size of UDP. Initially, DNS messages were limited to 512 bytes, a size that prevented IP fragmentation. Most standard network links have MTUs large enough to accommodate these DNS messages (considering an 8-byte UDP header and a 40-byte IPv6 header, resulting in a maximum payload size of 560 bytes for link layer frames). However, with the introduction of the Extension Mechanisms for DNS (EDNS0), this limit theoretically increased to 64 kilobytes. Using EDNS0, one can increase the size of DNS messages up to any k bytes, provided the MTU of the network link is greater than $k + 8 + 40$ bytes.

Thus, the optimum DNS message size to avoid IP fragmentation while minimizing the use of TCP will depend on the MTU of all the network links connecting two endpoints. Unfortunately, there is not yet a standard mechanism for DNS server implementors to access this information. Until such a standard exists, it is usually recommended that the EDNS0 buffer size should, by default, be set to a value small enough to avoid fragmentation on the majority of network links in use today. An EDNS0 buffer size of 1232 bytes will avoid fragmentation on nearly all current networks. This is based on an MTU of 1280, which is required by the IPv6 specification, minus 48 bytes for the IPv6 and UDP headers. Therefore, the currently recommended DNS message size over UDP is 1232 bytes.

EDNS0. Extension Mechanisms for DNS (EDNS0) [8] facilitates the transfer of DNS messages larger than 512 bytes. For this purpose, EDNS0 introduces a pseudo-record called OPT (*Options*) in the Additional section of a DNS message. Unlike traditional records, pseudo-records do not actually exist in a zone file and are created *on-the-fly*. In queries, a requester specifies the maximum UDP payload size it is capable of handling (known as EDNS0 buffer size) in OPT $\rightarrow$ `CLASS`. Additionally, the requester also indicates its support for DNSSEC by setting the `DO` (DNSSEC OK) bit in OPT $\rightarrow$ `TTL`.

OPT $\rightarrow$ `RDATA` also contains DNS cookies which provide a limited security against certain off-path attacks such as denial-of-service, cache poisoning, and answer forgery.

2.2 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 three[1] 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 records that have the same `NAME`, `CLASS` and `TYPE`. The resulting signature is stored in the `RDATA` field of an RRSIG record (consult Table 3).

2) DNSKEY. A DNSKEY record (refer Table 4) 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 everything else. When a resolver receives a DNS response with an RRSIG record, it uses the associated DNSKEY record to verify the signature contained therein.

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

Table 3: 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 RRs signed			
Signature Expiration	When the signature expires			
Signature Inception	When the records were signed			
Key Tag	ID of the key that can verify the sig.			
Signer's Name	Name of the signer			
Signature	sign(RRSIG $\longrightarrow$ RDATA‖RR(1)‖RR(2)‖ . . .) where RDATA excludes Signature and RR(i) is the i-th record in the RRset			

Table 4: DNSKEY **Wire Format**

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

Table 5: DS **Wire Format**

DS Record				
NAME	TYPE = DS	CLASS	TTL	RDLENGTH
RDATA				
Key Tag	ID of the KSK which is hashed			
Algorithm	Signature algorithm of the key			
Digest Type	Hash algorithm			
Digest	hash(DNSKEY $\longrightarrow$ NAME ‖ DNSKEY $\longrightarrow$ RDATA)			

3) Delegation Signer (DS). The DS record (Table 5) plays a vital role in establishing a secure chain of trust between parent and child zones. Whenever a resolver verifies RRSIGs using the ZSK_{pk} of a child, 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 aid resolvers in this endeavour, the child generates a hash of its KSK_{pk} and shares it with its parent in a DS record.

During a DNS lookup, when a resolver is referred to a child 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.

Table 6: RRFRAG **Wire Format**

RRFRAG Record				
NAME = (.)	TYPE = RRFRAG	CLASS = RRID	TTL = CURIDX	RDLENGTH = FRAGSIZE
RDATA				
RRSIZE	Specifies the size of the original resource record			
FRAGDATA	Contains the raw fragment bytes			

— NAME: must always be root (.)
— TYPE: identifies RRFRAG type
— Class: contains RRID identifying the particular resource record that is being fragmented
— TTL: contains CURIDX specifying the current index in the byte array of the original record which is being fragmented
— RDLENGTH: contains FRAGSIZE specifying the total number of bytes contained in RDATA

DNSSEC Lookup. This is similar to the DNS lookup described in §2.1, except that the resolver now sets the DO 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. Moreover, 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 $\longrightarrow$ AD set.

2.3 ARRF

DNSSEC works seamlessly when the size of the DNS message does not exceed the recommended UDP limit. However, post-quantum signature schemes have larger public key and signature sizes that cannot fit within these constraints.

To handle post-quantum cryptography in DNSSEC without relying on IP fragmentation or TCP fallback, a recent solution called ARRF was introduced by Goertzen and Stebila [11]. This approach conducts fragmentation at the application layer using a daemon running on top of the DNS software of resolvers and nameservers.

ARRF introduces a new type of DNS resource record called Resource Record Fragment (RRFRAG), similar to the existing pseudo-resource record OPT. RRFRAGs are not explicitly part of DNS zones; they are created when needed and use a generic DNS resource record wire format with some fields repurposed (see Table 6).

Whenever a DNS response size is too large to fit within the resolver's advertised UDP limit, RRFRAGs are used to split the resource records across multiple queries, ensuring that each response's size remains below the specified threshold.

RRFRAGs replace resource records in place, maintaining the original message format. However, the OPT resource record, containing important metadata like the DNS cookie, is not fragmented.

The initial response containing at least one RRFRAG acts as a *map* of the non-fragmented message. Requesters use this map to determine how to reassemble the original large DNS message. They can identify missing fragments and send new queries for those missing RRFRAGs. To specify which fragment they want, the size of those fragments, and their starting positions, requesters add RRFRAGs for each distinct RRID in the query's Additional section. When responders receive queries with RRFRAGs, they construct a standard DNS response by inserting the corresponding RRFRAGs into the Answer section. The FRAGDATA being sent is a copy of the desired resource record's bytes, starting at CURIDX and ending at CURIDX+FRAGSIZE. This request/response cycle continues until the requester successfully reassembles the original large message. Also, after receiving the initial response with the map, requesters can make subsequent RRFRAG requests in parallel.

Unfortunately, ARRF suffers from two major limitations, as acknowledged by its authors in [11]. Firstly, RRFRAG is a non-standard DNS record type, which can potentially cause middle boxes to reject the DNS message. Secondly, ARRF is vulnerable to memory exhaustion attacks. An on-path adversary can insert many RRFRAGs in the initial response with very large RRSIZEs. Since DNSSEC validation cannot take place until the full DNS message is reconstructed, the requester has no choice but to allocate memory for storing intermediate fragments. ARRF also needs a minimum of two round-trips to reconstruct the full DNS message. This is because the resolver requires RRIDs for fetching additional fragments. However, it only learns the RRIDs after receiving the initial response.

3 QNAME-Based Fragmentation (QBF)

We now present our solution for retro-fitting post-quantum cryptography in DNSSEC over UDP. Our scheme, called QNAME-Based Fragmentation (QBF), fragments and reassembles large DNS messages using a daemon running on top of the DNS software (here, BIND9) of resolvers and nameservers. The daemon intercepts and modifies (if necessary) all the incoming and outgoing DNS packets. It is also able to construct and send a DNS query of its own.

Unlike previous schemes, which fragmented the whole DNS message [24] or the DNS records [11], the QBF daemon fragments *only* the following: 1) Raw signature bytes stored in the field RRSIG $\longrightarrow$ RDATA $\longrightarrow$ Signature, and 2) Raw public key bytes stored in the field DNSKEY $\longrightarrow$ RDATA $\longrightarrow$ Public Key. Thus, QBF fragments *resemble* the original DNS response except insofar as they carry partial signatures or public keys. To fetch an additional fragment, the requester daemon constructs a new DNS query and provides information about the desired fragment in QUESTION $\longrightarrow$ QNAME.

QBF can be configured with the following modes: 1) Sequential: Wait to receive a requested fragment before sending another request; 2) Parallel 2-RTT: Request and receive the first fragment. Then request and receive all other fragments in parallel; and 3) Parallel 1-RTT: Request and receive all fragments in parallel. We now proceed to delineate QBF in the following subsections.

3.1 Fragment Wire Format

When a DNSSEC response size exceeds the requester's advertised EDNS0 buffer size, the responder daemon puts partial signature or public key bytes while keeping the rest of the message intact. We illustrate this by means of the following example scenario.

Consider a requester that sends a DNSSEC query asking for a Type A DNS record containing the IPv4 address of a domain test.example. Assume that the requester specifies a maximum EDNS0 buffer size of y bytes (*e.g.,* $y = 1232$ bytes, the recommended DNS message size over UDP) in OPT $\longrightarrow$ CLASS.

On the responder side, the generated DNS response is shown in Table 7 (Left). Assume that the size of this response is z bytes. The Answer section contains one Type A record holding the IPv4 address 1.2.3.4 for test.example and one RRSIG holding a signature over the former Type A record. The rest of the sections are empty (except for OPT in the Additional section).

The responder daemon observes that the response size exceeds the requester's UDP payload limit by 2 bytes (*i.e.* $z = y + 2$). Hence, it removes 2 bytes from RRSIG $\longrightarrow$ RDATA $\longrightarrow$ Signature (*i.e.* if RRSIG $\longrightarrow$ RDLENGTH is x bytes, then after removing 2 bytes from Signature, it becomes $x - 2$) and sets the HEADER $\longrightarrow$ TC flag to 1. The DNS response, referred to as *Fragment 1*, is displayed in Table 7 (Middle). Simultaneously, the responder daemon prepares another DNS response, known as *Fragment 2*, containing the remaining 2 bytes of the signature. At this juncture, the responder stores Fragment 2 in its cache. When the requester subsequently queries the additional fragment, the daemon retrieves it from its cache and transmits it back. Further details regarding the format of additional fragments are elaborated upon in the subsequent discussion. Note that the OPT record (which may hold important DNS cookies) remains intact after fragmentation.

Note that when a generated DNS response contains multiple RRSIG records, the QBF fragmenter removes an equal number of raw signature bytes from each RRSIG. A similar logic is applied in case of multiple DNSKEY records.

3.2 Requesting Fragments

After receiving Fragment 1, a requester daemon requests for additional fragments by constructing a new DNS query and setting QUESTION $\longrightarrow$ QNAME in the following format:

$$\langle\text{DELIMITER}\rangle\ \langle\text{fragnum}\rangle\ \langle\text{DELIMITER}\rangle\ \langle\text{domain}\rangle$$

DELIMITER is a non-valid domain name character such as ?, #, *, etc. The first delimiter indicates to the responder daemon that the DNS query is for a fragment. The second delimiter is used to mark the end of fragnum field since domain names can also start with a number. We use ? as DELIMITER throughout this paper. The fragnum is the desired fragment number and must be > 1 since Fragment 1 is received as response to the original DNS query. The domain is the domain name for which the fragment is required.

In the running example, to retrieve Fragment 2, the requester daemon sends a new DNS query with the QUESTION $\longrightarrow$ QNAME set to ?2?test.example (or in general, set to ?i?test.example for fetching the i-th fragment).

Original response	Fragment 1	Fragment 2
Header Section	**Header Section**	**Header Section**
Question Section	**Question Section**	**Question Section**
QNAME = `test.example`	QNAME = `test.example`	QNAME = `?2?test.example`
QTYPE = `A`	QTYPE = `A`	QTYPE = `A`
QCLASS	QCLASS	QCLASS
Answer Section	**Answer Section**	**Answer Section**
NAME	NAME	
TYPE = `A`	TYPE = `A`	
⋮	⋮	
RDLENGTH = 4	RDLENGTH = 4	
RDATA = `1.2.3.4`	RDATA = `1.2.3.4`	
NAME	NAME	NAME
TYPE = `RRSIG`	TYPE = `RRSIG`	TYPE = `RRSIG`
⋮	⋮	⋮
RDLENGTH = x	RDLENGTH = $x - 2$	RDLENGTH = $x - 2$
RDATA	RDATA	RDATA
Type Covered	Type Covered	Type Covered
Algorithm	Algorithm	Algorithm
⋮	⋮	⋮
Signer's Name	Signer's Name	Signer's Name
Signature = `0x3a4b5c6d`	Signature = `0x3a4b`	Signature = `0x5c6d`
Authority Section	**Authority Section**	**Authority Section**
Additional Section	**Additional Section**	**Additional Section**
OPT	OPT	OPT

Table 7: Wire format: Original response (Left), Fragment 1 (Middle), Fragment 2 (Right)

The resulting DNS response is shown in Table 7 (Right). This specific DNS response, referred to as Fragment 2, was originally prepared by the responder daemon during the fragmentation process and subsequently cached.

Note that the responder daemon sets HEADER → `TC` flag in all the fragments for maintaining backward compatibility. Furthermore, for all fragnum > 1, the responder daemon removes all records except RRSIG, DNSKEY and OPT. This is an efficiency measure to reclaim the space taken by the redundant records that were already sent in Fragment 1.

If a requester daemon sends a fragment query that cannot be answered (*e.g.,* a fragment with number fragnum is desired, but the entire message requires only (fragnum − 1) fragments), the responder daemon returns an error response with HEADER → `RCODE` set to `FORMERR` (indicating a problem with query format).

3.3 Execution Modes

We now describe the functionality of the QBF daemon in both the Parallel 2-RTT and Parallel 1-RTT modes.

We begin with Parallel 2-RTT, as it lays the foundation for the Parallel 1-RTT mode. These execution modes explain how a DNSSEC query initiated at a resolver is managed by the client-side QBF daemon deployed on the resolver and the server-side QBF daemon running on a DNS server. In this context, we assume the DNS server to be an Authoritative Name Server (ANS).

3.3.1 Parallel 2-RTT. Consider a scenario in which the resolver sends a DNSSEC query, requesting a Type `A` DNS resource record with QUESTION → `QNAME` set to `test.example`. The resolver, operating under the constraint of processing DNS responses of a recommended size, specifically, a maximum of 1232 bytes, configures the EDNS0 buffer size to 1232 within the OPT → `CLASS` field. We now describe how QBF operates in its Parallel 2-RTT mode to resolve the aforesaid query. The execution in Parallel 2-RTT proceeds as follows (Fig. 2 gives a schematic view of the QBF daemon operating in Parallel 2-RTT mode):

(1) The resolver daemon forwards the outgoing query *as it is.*
(2) Upon receiving the query, the QBF daemon on the ANS sets OPT → `CLASS` to a large value (here 65507, the maximum UDP payload size over IPv4) before forwarding the query to BIND9, the DNS server software. This value should be large enough to allow the retrieval of the full DNS response from BIND9 without truncation.
(3) The ANS daemon observes that the size of the full response exceeds 1232 bytes. Hence, it removes the requisite number of bytes from RRSIG → `RDATA` → `Signature` and marks the response as truncated (`TC`) before sending it to the resolver as Fragment 1. Note that from the complete BIND9 response, the ANS QBF daemon also prepares and caches all other fragments for a time interval (say, 5 seconds) within which it expects the resolver daemon to reach out for them.

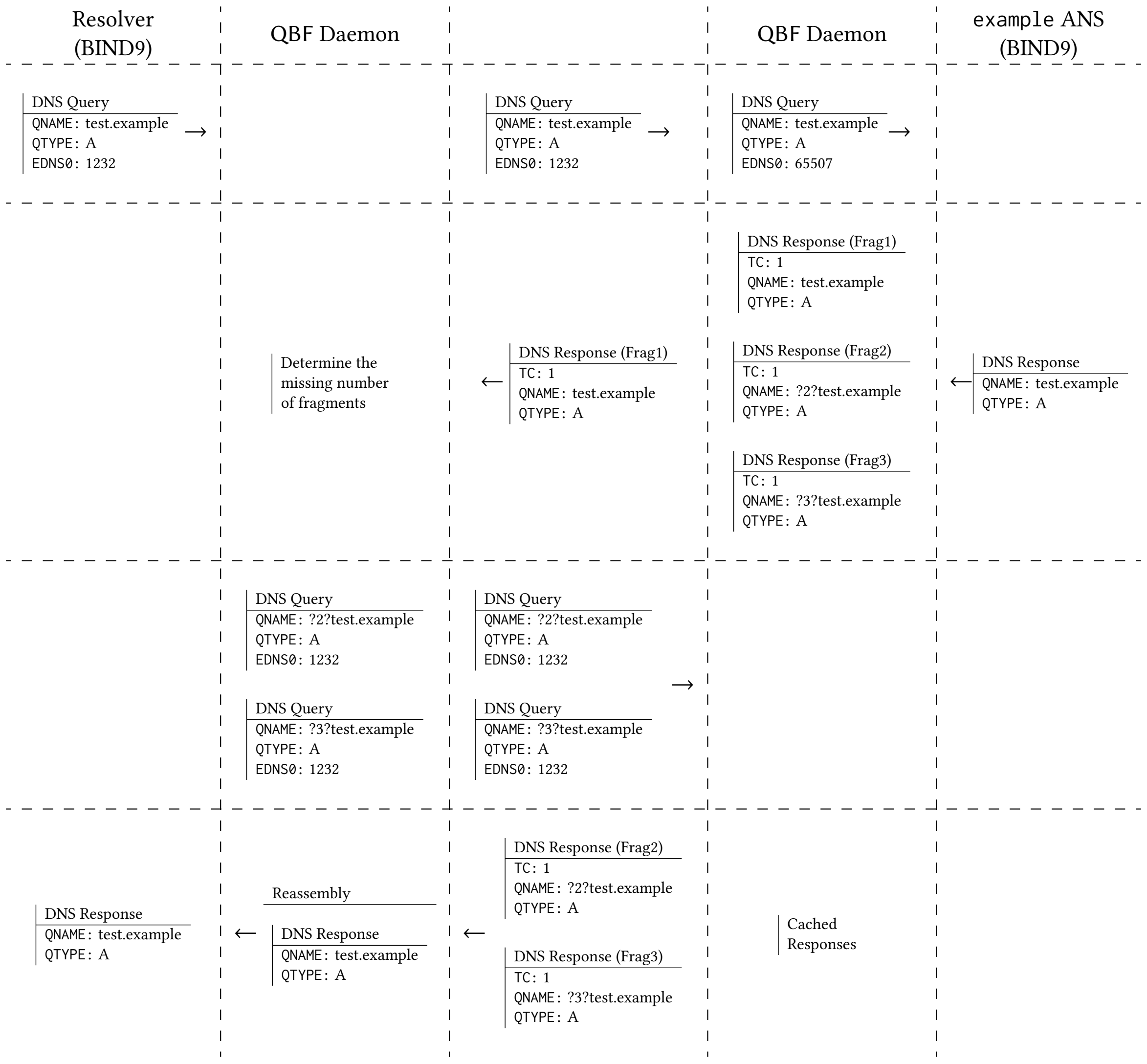

Figure 2: QBF **in Parallel 2-RTT mode**

In case, the ANS daemon does not receive any fragment requests from the resolver daemon within the time duration, it removes the fragments from its cache.

(4) On intercepting the first fragment, the resolver daemon calculates the required number of extra fragments as follows:
  (a) It retrieves the signature algorithm from RRSIG ⟶ RDATA ⟶ Algorithm and infers the size of a full signature (say, 690 bytes in case of Falcon-512).
  (b) It computes the size of the original DNS response by counting RRSIG ⟶ RDATA ⟶ Signature as having full size (690) instead of the partial size.
  (c) Subsequently, it calculates the number of additional fragments required based on the EDNS0 buffer limit.

(5) The resolver QBF daemon constructs the required number of extra fragment queries (2 in this case, see Fig. 2) in the format described in §3.2 and sends them in parallel.

(6) On the nameserver side, the QBF daemon responds to the fragment queries from its cache.

(7) On receiving all the fragments, the resolver QBF daemon appends RRSIG ⟶ RDATA ⟶ Signature from Fragment 2 and 3 (in sequence) to the corresponding place in Fragment 1. It then forwards the complete DNSSEC message to BIND9 for signature validation.

A similar process is followed for a response containing DNSKEY resource records. In that case, the relevant sections are DNSKEY ⟶ RDATA ⟶ Algorithm/Public Key.

3.3.2 Parallel 1-RTT. It is important to note that the resolver's QBF daemon has the flexibility to send additional fragment requests in parallel with the original query. The primary variable to consider is the precise number of these supplementary queries to dispatch. Should the daemon send more queries than necessary, it will simply receive a FORMERR response for each surplus query.

QTYPE	RR Type	Num. RRs	Falcon	Dilithium	SPHINCS$^+$
A	A	2	1*	6	22
	NS	1			
	RRSIG	3			
	OPT	1			
DNSKEY	RRSIG	2	2*	6	14
	DNSKEY	2			
	OPT	1			

Table 8: Number of additional queries required for QTYPE A DNSSEC query for `test.example` under a UDP constraint of 1232 bytes. Minimal responses and DNS cookies are disabled. * indicates that the last fragment has <100 bytes of free space.

Conversely, if the daemon sends an inadequate number of queries, it can always calculate the exact count of fragments based on the information provided in the first fragment (as elaborated in §3.3.1) and subsequently retrieve any remaining fragments.

Assuming that the daemon is cognizant of the nameserver's signing algorithm, possibly from prior interactions, it is feasible to estimate an upper-bound on the number of fragments. This estimation can be based on factors such as QUESTION → `QTYPE` and the EDNS0 UDP limit. In Table 8, we present the calculated number of extra queries needed in the running example of a QTYPE `A` DNSSEC query for `test.example`, for all NIST recommended post-quantum signature algorithms. Note that this represents a worst-case scenario (*i.e.* with *minimal-responses* disabled on the nameserver). For an overview of the structure of a non-minimal DNS response to a QTYPE A DNSSEC query, refer Table 9.

While the values in Table 8 will suffice for the majority of real-world DNSSEC queries, it is to be kept in mind that the total length of a domain name (*i.e.* label bytes and label length bytes) can be up to 255. Considerations also have to be made for QTYPE `AAAA` queries since the size of an IPv6 address is 4× the size of an IPv4 address. Additionally, a joint client-server DNS cookie in OPT → `RDATA` can occupy up to 40 bytes of space.

Fortunately, the resolver *a priori* knows the length of the domain name and the type of query from the `QNAME` and `QTYPE` fields of the DNS query, respectively, and the decision to use DNS cookies also rests with the resolver (*i.e.* a server inserts its cookie only if the client has provided its own). Thus, the resolver daemon should account for longer than average domain names, QTYPE `AAAA` IPv6 queries and usage of DNS cookies when determining the number of additional queries. If deemed necessary, it should increment the numbers marked with * in Table 8 by 1.

Figure 3 gives a schematic view of the QBF daemon operating in Parallel 1-RTT mode. Ideally, the resolver daemon should send the additional queries after a slight delay from the original one so as to give sufficient time to the responder daemon for preparing the fragment cache. Alternatively, the responder daemon should forward only the original query to the BIND9 software and prepare the cache from the resulting response as shown in Figure 3.

3.4 Backward Compatibility

We now discuss what happens when one of the end points implements QBF while the other one does not.

- QBF-unaware Requester | QBF-aware Responder: On receiving the first fragment, the requester will detect that the HEADER → `TC` flag is set. It will then discard this response and retry the query over TCP.
- QBF-aware Requester | QBF-unaware Responder: If the requester daemon sends queries in Parallel 1-RTT mode, it will receive the first response with `TC` flag set and the rest of the responses with HEADER → `RCODE` set to `NXDOMAIN` (Non-Existent Domain). It will then infer that the responder does not support QBF and repeat the query over TCP.
  In case the requester daemon is running in Sequential or Parallel 2-RTT mode, it will receive the initial response with `TC` flag set. However, responses truncated by the DNS software only contain the HEADER section, the Question section and an OPT record in the Additional section (*i.e.* the response is identical to the query but with `TC` set). On detecting this, the daemon will conclude that the responder is QBF-unaware and fallback to TCP.

3.5 Security Considerations

Cache Poisoning. Since the daemon forwards the complete response to the resolver for DNNSEC validation, DNS cache poisoning is not a concern assuming a secure algorithm is used for signing.

UDP Unreliability. Since QBF messages (requests or responses) ultimately travel over UDP, it is possible that some messages may fail to reach their destination. The two immediate solutions for this would be: 1) If the resolver does not get a response from its QBF daemon within 800 ms (the default BIND9 timeout), it sends a fresh query again and the whole process starts over; 2) If the QBF daemon does not receive a fragment response within a shorter timeout (say, 100 ms), it re-sends the fragment query.

DNSSEC Downgrade Attacks. Heftrig *et al.* [12] found that 45% of DNS resolvers do not perform DNSSEC validation when a new signing algorithm is used in the DNS responses. Therefore, unless IETF standardizes the correct behaviour of resolvers in such a scenario, this vulnerability can possibly continue to affect DNSSEC when post-quantum algorithms are deployed.

Denial of Service (DoS). *Off-path attacks:* Because each fragment has to be explicitly requested for, a requester daemon can reject any unexpected fragment it receives. Thus, with the aid of DNS cookies, off-path attacks are rendered infeasible. *On-path attacks:* If an adversary or a malicious middlebox tampers with the data in the resource records, it'll eventually cause DNSSEC validation to fail on the resolver's end.

Memory Exhaustion Attacks. An on-path adversary cannot cause a QBF requester daemon to allocate an arbitrarily large amount of memory for fragments. Recall that Fragment 1 is identical to the original DNS response, except that it carries only partial raw signature or public key bytes. If parallel 2-RTT or sequential mode is being used, the only way the adversary can cause the requester daemon to over-compute the number of additional fragments is 1) by changing `RDATA` → `Algorithm` or 2) by inserting many resource records containing fewer raw signature or public key bytes.

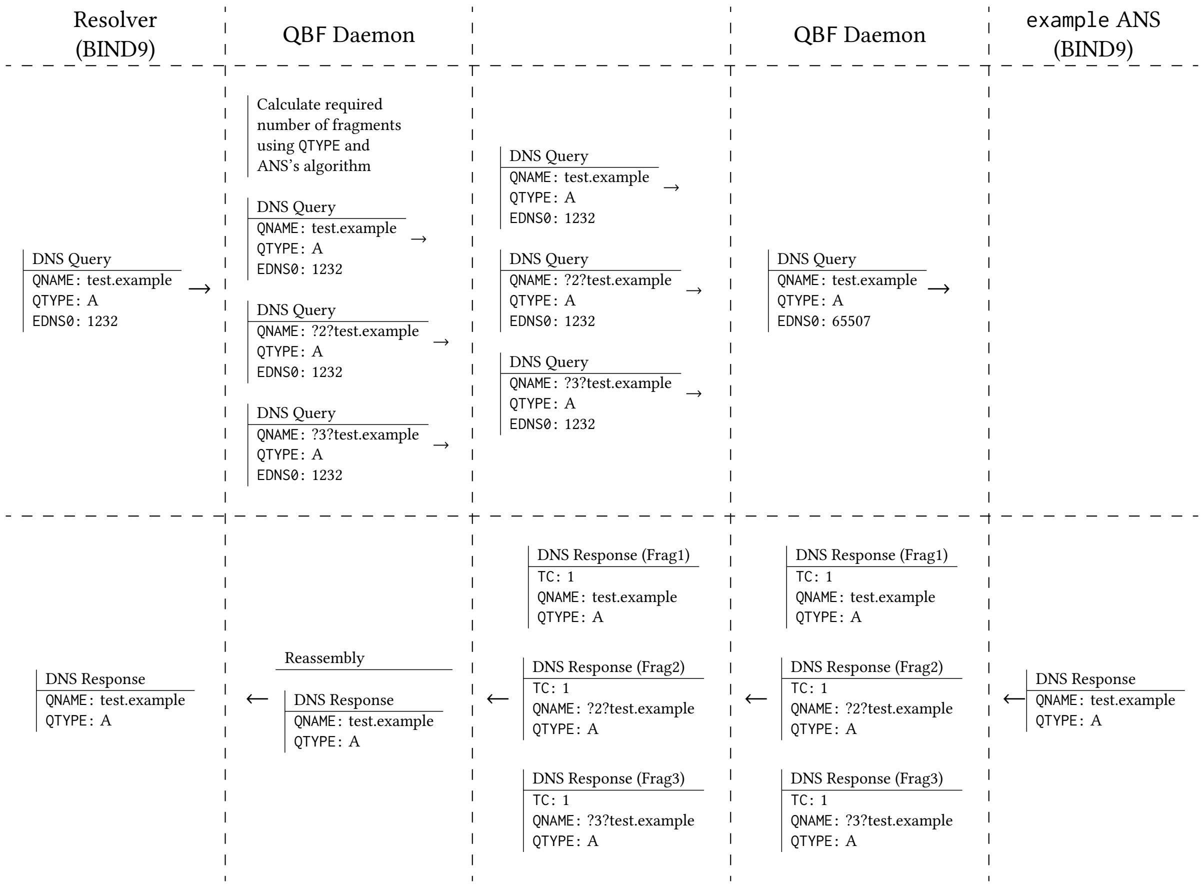

Figure 3: QBF **in Parallel 1-RTT mode**

In the first attack, the adversary is limited to changing the algorithm to the one with the largest signature or public key footprint (For example, SPHINCS$^+$ which has 7856 bytes of signature). On the other hand, the resolver daemon can easily detect the second type of attack based on the response it is expecting for a particular QTYPE. For example, even in the worst-case, a QTYPE A/AAAA DNS response cannot have more than three[1] RRSIGs as shown in Table 8. Note that in Parallel 1-RTT, the number of fragments (and hence, the memory allocated) is fixed as discussed in §3.3.2.

4 Evaluation

In this section, we provide implementation details of QBF and compare its DNS resolution performance with Standard DNS with TCP fallback and Parallel ARRF [11].

4.1 Setup

We use the source code of ARRF [11] as base to build the QBF daemon. The DNS software is BIND 9.17.12. The cryptographic stack is `openssl` 1.1.1 and `liboqs` 0.7.2 [26]. The daemon is written in C and uses `libnetfilter-queue` 1.0.5-2 to intercept incoming and outgoing DNS packets.

[1]If multiple NS records are sent in the Authority section, then the Additional section can contain multiple RRSIGs. However, the count thereof is generally 2-3.

Docker 4.22 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`. All experiments are run on a MacBook Air M1 with 8 GB of RAM.

4.2 Network Setting

We design a DNS network with 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 is running as a private Ubuntu 22.04 Docker container having a networking constraint of 50 Mbps bandwidth and 10 ms latency. Additionally, the resolver is configured with `send-cookie no;` in its `named.conf`.

For simplicity, each zone is signed with a single algorithm and has one ZSK and one KSK. The zone file of ANS contains 10 Type A records, each with a unique domain name and an associated RRSIG. The ANS is configured with `minimal-responses no;` in its `named.conf` which means that it will be as complete as possible when generating responses. This represents the worst-case scenario in terms of response size. Table 9 illustrates the format of non-minimal responses generated by the ANS's BIND9 software.

Header Section
Question Section
Answer Section
Type A RR
RRSIG
Authority Section
Type NS RR
RRSIG
Additional Section
Type A RR
RRSIG
OPT

Header Section
Question Section
Answer Section
DNSKEY
DNSKEY
RRSIG
RRSIG
Authority Section
Additional Section
OPT

Table 9: Structure of non-minimal DNS responses to a QTYPE `A` query (left) and to a QTYPE `DNSKEY` query (right)

Table 10: Average resolution time (±1 ms) of 10 QTYPE `A` queries in a 10 ms latency and 50 Mbps network setting. * indicates a TCP fallback. The parallel variants of ARRF and QBF show high scalability under growing signature sizes because of sending the fragment requests in parallel.

Algorithm	Standard DNS	ARRF (Parallel)	QBF (2-RTT)	QBF (1-RTT)
ECDSA-P256	42	-	-	-
RSA-2048	42	-	-	-
Falcon-512	83*	63	63	43
Dilithium2	83*	64	64	44
SPHINCS$^+$-128s	85*	65	66	46

4.3 Experiment and Results

In this experiment, we measure the average resolution time of QTYPE `A` DNSSEC queries when the resolver already has DNSKEY and NS records of the nameservers. In such a case, the resolver directly contacts the ANS with QTYPE `A` queries. The results of this experiment for the three NIST recommended signature algorithms [2]: Falcon-512 at security level 1, Dilithium2 at level 2, and SPHINCS$^+$ at level 1 are tabulated in Table 10.

All parallel variants of QBF yield substantially lower resolution times than Standard DNS for post-quantum DNSSEC queries. More concretely, QBF in 1-RTT and 2-RTT mode is approximately 50% and 25% faster than Standard DNS, respectively. This is because DNS, as standardized, incurs the penalty of 1) the (initial) wasted trip over UDP and 2) the ensuing 3-way TCP handshake.

On the other hand, QBF in 1-RTT mode shows an improvement of about 30% over parallel ARRF because of requiring only 1 round-trip compared to two of the latter.

[2]Higher security levels are currently not supported by the OQS-BIND9 fork.

5 Conclusion

In this paper, we presented `QNAME`-Based Fragmentation (QBF): a backward-compatible solution for integrating post-quantum cryptography into DNSSEC over UDP. QBF achieves its goal using *only* standard DNS records and requires a *single* round-trip to reconstruct the full DNS message. We developed the QBF daemon which operates atop the DNS software of resolvers/nameservers. It manages the fragmentation/reassembly of large DNS messages without requiring any changes to DNS stack or zone files. Our experiments show that in resolving QTYPE `A` queries, QBF is about 50% and 30% faster than Standard DNS and Parallel ARRF, respectively.

Acknowledgments

Aditya Singh Rawat was supported by a research grant from the MPhasis F1 Foundation.

References

[1] [n. d.]. 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] [n. d.]. DNS Flag Day 2020. URL: https://www.dnsflagday.net/2020/. Accessed: 2023-06-14.

[3] [n. d.]. Is large-scale DNS over TCP practical? https://ripe76.ripe.net/presentations/95-jonglez-dns-tcp-ripe76.pdf. Accessed: 2024-07-09.

[4] Derek Atkins and Rob Austein. 2004. Threat Analysis of the Domain Name System (DNS). RFC 3833.

[5] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. 2019. The SPHINCS+ Signature Framework. In *SIGSAC CCS*.

[6] Ron Bonica, Fred Baker, Geoff Huston, Bob Hinden, Ole Trøan, and Fernando Gont. 2020. IP Fragmentation Considered Fragile. RFC 8900.

[7] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. 2018. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *EuroS&P*.

[8] Joao da Silva Damas, Michael Graff, and Paul A. Vixie. [n. d.]. Extension Mechanisms for DNS (EDNS(0)). RFC 6891.

[9] Pratyush Dikshit, Mike Kosek, Nils Faulhaber, Jayasree Sengupta, and Vaibhav Bajpai. 2024. Evaluating DNS Resiliency and Responsiveness With Truncation, Fragmentation & DoTCP Fallback. *IEEE TNSM* (2024).

[10] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR TCHES* (2018).

[11] Jason Goertzen and Douglas Stebila. 2023. Post-Quantum Signatures in DNSSEC via Request-Based Fragmentation. In *PQCrypto*.

[12] Elias Heftrig, Haya Shulman, and Michael Waidner. 2022. Poster: The Unintended Consequences of Algorithm Agility in DNSSEC. In *SIGSAC CCS*.

[13] Paul E. Hoffman and Patrick McManus. 2018. DNS Queries over HTTPS (DoH). RFC 8484.

[14] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. 2016. Specification for DNS over Transport Layer Security (TLS). RFC 7858.

[15] Christian Huitema, Sara Dickinson, and Allison Mankin. 2022. DNS over Dedicated QUIC Connections. RFC 9250.

[16] Panos Kampanakis and Tancrède Lepoint. 2023. Vision Paper: Do We Need to Change Some Things? - Open Questions Posed by the Upcoming Post-quantum Migration to Existing Standards and Deployments. In *SSR*.

[17] Mike Kosek, Trinh Viet Doan, Simon Huber, and Vaibhav Bajpai. 2022. Measuring DNS over TCP in the era of increasing DNS response sizes: a view from the edge. *SIGCOMM CCR* (2022).

[18] Jiarun Mao, Michael Rabinovich, and Kyle Schomp. 2022. Assessing Support for DNS-over-TCP in the Wild. In *PAM*.

[19] Moritz Müller, Jins Jong, Maran Heesch, Benno Overeinder, and Roland Rijswijk-Deij. 2020. Retrofitting post-quantum cryptography in internet protocols: a case study of DNSSEC. *SIGCOMM CCR* (2020).

[20] T. Prest, P.A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. 2022. FALCON. Tech. rep., National Institute of Standards and Technology (2022). *Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022* (2022).

[21] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. DNS Security Introduction and Requirements. RFC 4033.

[22] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. Protocol Modifications for the DNS Security Extensions. RFC 4035.

[23] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. Resource Records for the DNS Security Extensions. RFC 4034.

[24] Mukund Sivaraman, Shane Kerr, and Linjian Song. [n. d.]. DNS message fragments. https://datatracker.ietf.org/doc/draft-muks-dns-message-fragments/00/.

[25] Linjian Song and Shengling Wang. [n. d.]. ATR: Additional Truncation Response for Large DNS Response. https://datatracker.ietf.org/doc/draft-song-atr-large-resp/03/.

[26] Douglas Stebila and Michele Mosca. 2016. Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In *SAC (Lecture Notes in Computer Science, Vol. 10532)*. Springer, 14–37.

[27] Gijs Van Den Broek, Roland Van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. 2014. DNSSEC meets real world: dealing with unreachability caused by fragmentation. *IEEE Communications Magazine* (2014).

[28] Roland van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. 2016. A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. *IEEE JSAC* (2016).