Impact of Merkle Tree Ladder (MTL) Mode Signatures on DNSSEC

Jannik Peters
Security and Network Engineering
University of Amsterdam
jannik.peters@os3.nl

***Abstract*—Quantum computing is expected to threaten current cryptography, especially the algorithms used in many Internet protocols. Quantum-resilient algorithms, colloquially referred to as Post-Quantum Cryptography (PQC), are under active development and standardization. Many of these new algorithms have larger signatures or keys that exceed the size requirements imposed by the Domain Name System (DNS) or require increased computational power. This means that PQC algorithms need to be evaluated carefully for their use in the Domain Name System Security Extensions (DNSSEC). Previous work analyzed different PQC algorithms and found potential candidates for use in DNSSEC. To enable the use of current PQC algorithms with large signatures in DNSSEC, researchers devised a way to reduce the per-message impact of such algorithms by using a Merkle tree ladder (MTL) to authenticate messages, and only signing this data structure with the underlying PQC algorithm. This way all messages use a condensed signature that can be used alongside the Merkle tree ladder to authenticate a message. The ladder can be distributed out-of-band or attached to any condensed signature, turning it into a full signature.**

This project analyzes the impact of using MTL mode signatures in DNSSEC, by measuring the signing and verification performance, and the key and signature sizes, and by comparing the algorithms of the MTL mode reference implementation based on SLH-DSA to the currently deployed digital signature algorithm ECDSA Curve P-256 with SHA-256 and to other PQC algorithms analyzed in other projects. We find that the MTL mode signatures perform adequately well and provide condensed signatures small enough to meet DNS limitations. We find the proposed MTL mode signatures to be promising for use in DNSSEC, but that they could benefit from modifications to the DNS protocol, like an Extension Mechanisms for DNS (EDNS(0)) option to indicate an available ladder version, or by removing the SOA Resource Record (RR) from denial of existence responses.

***Index Terms*—DNS, Post-Quantum Cryptography, Security**

I. Introduction

Quantum computing is expected to threaten the security of our current cryptosystems in the next 16 years [1], [2]. To prepare for the time that quantum computers are feasible to build, cryptographers are already designing new cryptography algorithms that are resilient to cryptanalysis with both quantum computers and normal computers [3], colloquially referred to as Post-Quantum Cryptography (PQC). Some of the algorithms submitted during the competition of the National Institute of Standards and Technology (NIST) are already standardized or in the process of standardization [4].

The new cryptography algorithms need to replace existing ones in e.g. Internet protocols. However, some protocols, like the Domain Name System (DNS), have certain limitations that impose requirements on signature and key size, and signing and verification performance on the Post-Quantum Cryptography (PQC) algorithms usable for the Domain Name System Security Extensions (DNSSEC). Many of the new algorithms do not meet these requirements [5], [6], often because of their large signatures that do not fit into a single non-fragmented UDP packet (the standard transport for DNS) with the recommended maximum payload size of 1232 bytes [7].

To address this issue, J. Harvey *et al.* [8] devised a way to reduce the impact of the current PQC signature schemes by using Merkle tree ladders (MTLs). A technique where a collection of binary trees store the hashes of the messages to authenticate, and the underlying PQC signature scheme is only used to sign a data structure, the Merkle tree ladder, derived from this collection of binary trees. To authenticate a message, an authentication path into the tree containing the hash of the message is attached to the message as a condensed signature, which can be way smaller than the signatures of the underlying PQC algorithm. For their initial implementation in DNSSEC, J. Harvey *et al.* selected the Stateless Hash-Based Digital Signature Algorithm (SLH-DSA) (based on SPHINCS+ [9]), as it benefits the most from MTL mode, due to its large signatures, and is built on well-understood and proven cryptographic techniques, being hash-based instead of e.g. lattice-based.

In this project we measure the impact of using SLH-DSA in MTL mode (SLH-DSA-MTL) on DNSSEC, compare the results to other research measuring different PQC algorithms [5], and analyze its feasibility for DNSSEC.

II. Research Questions

RQ1. What is the impact of using SLH-DSA-MTL in DNSSEC?

To answer the main research question, we define the following sub-questions:

RQ2. How does the MTL mode affect the signature and key size of SLH-DSA?

RQ3. How does SLH-DSA-MTL signature and key size, and signing and verification performance compare to other PQC algorithms in the context of DNSSEC?

RQ4. How does SLH-DSA-MTL signature and key size, and signing and verification performance compare to the currently in DNSSEC deployed algorithm ECDSA Curve P-256 with SHA-256?

III. Related Work

M. Müller *et al.* [6] presented an analysis of the implications of different PQC algorithms in the context of DNSSEC. The authors specifically considered the requirements on signature size and verification efficiency as imposed by DNSSEC, and evaluated the (then current) PQC algorithms of the third round of the NIST competition. They show three algorithms that partially meet DNSSEC's requirements and show options to adapt DNSSEC to accommodate PQC algorithms better.

Following further progress in the NIST competition, C. Schutijser *et al.* [10] developed a testbed to easily investigate the impact of the PQC algorithms on DNSSEC in a local environment.

Additionally, J. Goertzen *et al.* [11] investigated the real-world impact of a few PQC algorithms on correct delivery of DNS messages via UDP with the RIPE Atlas measuring system [12] and show that most PQC algorithms would currently have significant delivery failure rates.

Recently, O. Surý [5] conducted performance benchmarks for a list of algorithms implemented in BIND 9 [13] and found that some algorithms provided inadequate (signing or verification) performance or signature or key sizes. In the benchmarks, O. Surý justifiably omitted the algorithm SLH-DSA.

Considering that the new PQC algorithms have rather large signatures and that, if there were to be future algorithms with smaller signature sizes, they would be less researched, J. Harvey *et al.* [8] devised a way to reduce the impact of the current PQC signature schemes by using Merkle tree ladders. They published two IETF Internet-Drafts specifying the *Merkle Tree Ladder (MTL) Mode Signatures* [8] and the use of SLH-DSA-MTL in DNSSEC in *Stateless Hash-Based Signatures in Merkle Tree Ladder Mode (SLH-DSA-MTL) for DNSSEC* [14].

Related to the research are the standardization documents FIPS 204 [15] and 205 (SLH-DSA) [16], and the participants of the different rounds of the NIST competition [4].

This project combines and continues the work of O. Surý and A. Fregly *et al.*, by conducting performance benchmarks for the SLH-DSA-MTL algorithm in DNSSEC.

IV. Background

This report assumes basic knowledge about the Domain Name System, DNSSEC, and its underlying protocols. More specifically, topics like the root zone, DNS messages, the origin of the 1232 bytes UDP payload limitation [7], [17], and others, will not be explained here.

Additionally, basic knowledge about CPU cores/threads and hyper-threading is beneficial to fully grasp the reasoning behind the benchmark environment setup in Section VII, but is not required to follow the rest of the paper.

Basic knowledge about using cryptography in practice is necessary, as concepts like signatures, signature creation, and signature verification are not further explained. Understanding of the inner workings of cryptography algorithms is not required.

V. Introduction to MTL mode

This section is primarily a brief and simplified recapitulation of the Merkle tree ladder mode description from [18] and [14]. For a more in-depth explanation, please refer to those resources. Additionally, this description uses DNS and DNSSEC specific terms instead of general cryptography terms.

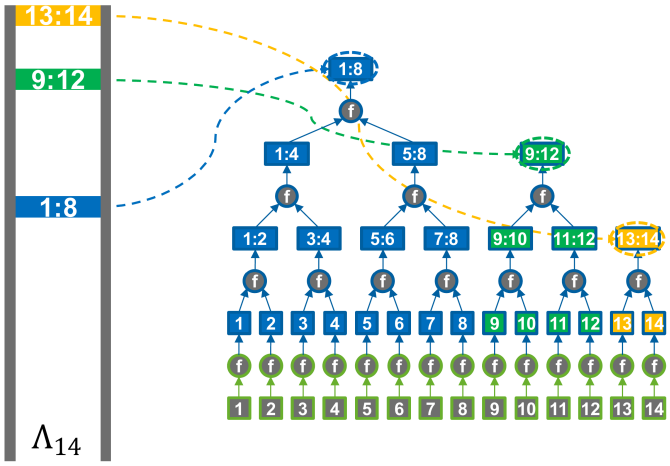

Fig. 1. "Example of a Merkle tree ladder following a binary rung strategy. Rungs [1:8], [9:12] and [13:14] collectively authenticate all 14 leaf nodes." [18]

In normal DNS operation, a digital signature algorithm is used to sign each individual Resource Record set (RRset). In MTL mode, however, RRsets are authenticated using a data structure (the Merkle tree ladder) derived from all RRsets in a zone, which in turn is signed with the underlying signature scheme. Individual RRsets are then accompanied by condensed signatures (see below).

The Merkle tree ladder is a collection of binary trees, or rather their root nodes. In the binary rung strategy, provided by the authors, the number of binary trees and the number of messages they cover each is determined by the binary representation of the number of RRsets, so, e.g. with 14 RRsets (as in Figure 1), which is $8 + 4 + 2$, so 3 trees. The 14 RRsets in this example are then laid out as the leafs of those binary trees, or rather their hash values are (see bottom two rows in Figure 1). The internal nodes of the binary trees are constructed from the hash values of their two children, all the way up to the root. The root node of each binary tree is called a rung, and the collection of rungs is called a ladder.

There are two types of signatures in MTL mode, full signatures and condensed signatures. Full signatures consist of the ladder, signed with the underlying signature algorithm, and the authentication path (see below) for the RRset in question. Condensed signatures consist of the authentication path and some metadata to identify the ladder from which the authentication path was formed.

An authentication path is a list of sibling node hashes on the path from the leaf of an RRset to the rung of its binary tree. So, e.g. for message 2 in Figure 1, the siblings would be the leaf node of message 1, the node [3:4], and [5:8].

By combining the sibling node hash of message 1 with the hash of the original message 2, a verifier can reconstruct node [1:2]. Next, the verifier can combine the nodes [1:2] and [3:4] to get [1:4], which is then combined with [5:8] to calculate the root node [1:8]. If this calculated root node is part of the ladder that was distributed as part of any full signature and stored by the verifier, and its signature is valid, then the message is authenticated.

In the current version of the Internet-Draft about the use of SLH-DSA in MTL mode [14] the SOA RR is always accompanied by a full signature while other RRs use condensed signatures. However, every RRset could be served with a full signature. A client can request to receive a full signature for any query by using the EDNS(0) flag specified in the draft.

VI. Approach & Methods

To better measure the performance of SLH-DSA-MTL, we prepare a consistent benchmark environment, assigning a specific set of CPU cores to the benchmarking processes (see Section VII), to decrease the performance variance due to interfering processes. With the benchmark environment set up, we create a performance baseline using ECDSA Curve P-256 with SHA-256 (ECDSAP256SHA256) [19], both because it is featured in [5] and because it is a required algorithm as of RFC 8624 [20]. Next, we benchmark the SLH-DSA-MTL algorithm variants implemented in [21], and compare their performance against the ECDSAP256SHA256 baseline. We analyze key generation, signing, and verification performance, as well as key and signature size.

To compare our results to the results from [5], we derive a performance ratio based on the baseline algorithm ECDSA that has been measured by us and by O. Surý [5]. We adjust the measurements in [5] by this ratio to be able to compare our measurements to the measurements from [5] even though we used different software and hardware.

VII. Environment Setup

A. Isolating CPU cores

To reserve 4 thread (2 cores) for the benchmarks, we use the kernel command line option `isolcpus=12-15` [22]. This instructs the Linux kernel process scheduler not to schedule any processes on the threads 12–15.

B. Running benchmarks on isolated cores

To execute processes on the isolated CPU cores, we use `taskset` [23], specifying the previously reserved threads.

Even though we reserve multiple threads, the benchmarks only use a single thread, as the `ldns` tools are implemented single-threaded.

C. Benchmark and measurement tool

To execute and collect statistics during the benchmarks, we use `hyperfine` version 1.19.0 [24]. This tool allows us to execute the programs to benchmark for a set number of times, collect each execution's time, and calculate relevant statistics, such as mean, median, and variance.

D. Disabling CPU boost

During initial tests, we observed a relatively stable clock speed (of ca. 4600–4750 MHz) and no notable performance variance with CPU boost enabled. However, the CPU temperature would rise to 99 °C during the single-threaded benchmarks, and we decided to disable CPU boosting to avoid potential thermal throttling, in case other CPU cores were utilized during the benchmarks.

On our machine, disabling the CPU boost can be achieved by writing to the Linux `sysfs`, namely the files `/sys/devices/system/cpu/cpu{0..15}/cpufreq/boost`.

With CPU boost disabled, the clock speed during benchmarks is very stable at 3269 MHz with a CPU temperature around 55 °C.

E. Other system preparations

In addition to isolating CPU cores, we mount a `tmpfs` to store all files created during the benchmarks in system memory to avoid any potential performance impact by the disk.

As the machine used for the benchmarks is a laptop with a desktop operating system, we also disable the system's file indexing for the benchmark directory to avoid any unnecessary I/O on said directory that could interfere with the results.

F. Resolver setup

To measure the message sizes with the different algorithms as shown in [5], we set up an authoritative name server to serve the root zone signed with the different algorithms, and a resolver (that only queries our own name server). The software used are modified versions of NSD [25] and Unbound [26] extended to support the two MTL mode algorithms.

G. Preparing the root zone file

For our experiments we use the root zone [27], as it is easy to acquire (e.g. from the root server `f`), very relevant, and frequently queried. We remove all DNSSEC related records that will be re-created by the signing process. Additionally, we remove the record for the *Message Digest for DNS Zones* [28], as it will contain a wrong digest after our modifications to the zone and is irrelevant for our experiments. In summary, we remove records of the types `DNSKEY`, `NSEC`, `RRSIG`, and `ZONEMD`:

```
dig AXFR . @f.root-servers.net \
  > root-original.zone
grep -vE "(DNSKEY|NSEC|RRSIG|ZONEMD)" \
  root-original.zone | \
  grep -vE "(^;)|(^$)" | \
  head -n-1 > root.zone
```

Listing 1. Steps taken to prepare the root zone.

At the time of our experiments, the root zone's serial number is: 2025060900. With a total of 20821 RRs and 2787 RRsets. This is the version used throughout this research.

VIII. The Benchmarks

For the benchmarks, we use the extended LDNS tools [21] for key generation, and signature creation and verification. The key size can be derived from the generated key files and read from the algorithm standardization documents (e.g. [16]). The signature sizes need to be extracted from the RRSIG records of the signed zone file. Additionally, we can calculate the maximum signature size (full and condensed) using the formulas provided in [14, Section 11].

To create an adequate sample size for each benchmark, we repeatedly execute the different programs used for key generation, zone signing and zone verification, and measure their execution time[1]:

TABLE I
Benchmark runs per algorithm.

Algorithm	Key Generation	Signing	Validation
ECDSAP256SHA256	10000	1000	1000
SLH-DSA-MTL-SHA2-128s	10000	1000	1000
SLH-DSA-MTL-SHAKE-128s	10000	1000	1000
SLH-DSA-SHA2-128s	1000	3	100
SLH-DSA-SHAKE-128s	1000	3	100

To facilitate reproducable and quick benchmarks, we developed a simple script to prepare, run, and clean up the benchmarks, as well as logging CPU clock speeds during the benchmarks. It executes each benchmark on the reserved threads using the following command where `$cmd` is replaced by the LDNS invocation for each benchmark:

```
taskset --cpu-list 12-15 \
    hyperfine \
        --export-json="$json" \
        --shell=none \
        --warmup "$warmup" \
        --runs "$runs" \
        -- \
        "$cmd"
```

Listing 2. Command to run a benchmark on specific CPU threads.

A. Key generation

For key generation, we use `ldns-keygen -a <alg> .` for each algorithm.

B. Zone signing

For zone signing, we use `ldns-signzone -f <output-file> root.zone <zsk> <ksk>` with two keys (one ZSK and one KSK) for each algorithm.

We use one KSK and one ZSK for signing the zone, to be in line with [5] and operator procedures. However, this does not influence the benchmarks significantly, compared to only using a single key, as the additional key is only used to sign a single RRset, which amounts to 0.036% of the signatures created for the root zone (and an even smaller share for larger zones).

C. Zone verification

For zone verification, we use `ldns-verify-zone -k <ksk-file> <signed-zone>` for each algorithm, where the zone is signed using one ZSK and one KSK as above for the zone signing benchmark.

D. Calculating the maximum signature size

A. Fregly *et al.* [14] provide formulas to calculate the maximum signature size per number of RRsets in a zone with input parameters:

- n = Security parameter for the underlying signature scheme
- USS = Size of underlying signature
- N = Number of messages in message series

In our case with the algorithms SLH-DSA-SHA2-128s and SLH-DSA-SHAKE-128s the values for n and USS are $n = 16$ and USS $= 7856$.

To calculate the maximum condensed signature size, the formula is:

$$C = n + 24 + (n * \text{floor}(\log2(N))) \quad (1)$$

To calculate the maximum size for a signed ladder, we use:

$$L = 16 + ((8 + n) * \text{ceiling}(\log2(N))) + \text{USS} \quad (2)$$

Finally, the maximum size for full signatures can be calculated using:

$$F = C + L \quad (3)$$

IX. Results

In this section we present the results of the signing and verification performance benchmarks, the key and signature size benchmarks, the maximum signature size calculation, and the measurements from [5] adjusted by the resulting performance ratios.

A. Signing performance

MTL mode signatures reduce the signing time of the underlying signature scheme SLH-DSA by a factor of 666.30 for SHA2 and 894.0 for SHAKE down to 598.52 ms and 902.96 ms respectively. This is 1.67 and 2.52 times the time required for ECDSAP256SHA256, but still within one second.

TABLE II
Signing time in milliseconds.

Algorithm	Mean	σ
ECDSAP256SHA256	358.17	8.64
SLH-DSA-MTL-SHA2-128s	598.52	10.03
SLH-DSA-MTL-SHAKE-128s	902.96	5.79
SLH-DSA-SHA2-128s	398 793.40	776.08
SLH-DSA-SHAKE-128s	807 239.86	762.97

[1]Initially, we planned to run the non-MTL mode SLH-DSA variants as often as the other algorithms, but found that they are too expensive and reduced the number of runs.

B. Verification performance

MTL mode signatures also reduce the verification time of the underlying signature scheme SLH-DSA by a factor of 4.96 for SHA2 and 5.79 for SHAKE down to 598.06 ms and 600.89 ms respectively. This is, in both cases, approximately 1.09 times the time required for ECDSAP256SHA256, but also within one second.

TABLE III
VERIFICATION TIME IN MILLISECONDS.

Algorithm	Mean	σ
ECDSAP256SHA256	550.19	4.52
SLH-DSA-MTL-SHA2-128s	598.06	8.40
SLH-DSA-MTL-SHAKE-128s	600.89	7.63
SLH-DSA-SHA2-128s	2968.52	54.94
SLH-DSA-SHAKE-128s	3476.51	22.02

C. Key generation performance and size

While the key generation for the different SLH-DSA variants takes over 7 or even 14 times the amount of time as for ECDSAP256SHA256, keys generate humanly imperceptibly fast for all algorithms:

TABLE IV
KEY GENERATION TIME IN MILLISECONDS.

Algorithm	Mean	σ
ECDSAP256SHA256	2.96	0.18
SLH-DSA-MTL-SHA2-128s	22.37	0.79
SLH-DSA-MTL-SHAKE-128s	41.83	1.03
SLH-DSA-SHA2-128s	22.20	0.61
SLH-DSA-SHAKE-128s	41.85	0.68

The public keys of the SLH-DSA variants are all 32 bytes in size, which is also documented in [14, Section 3] and [16, Table 2]. This is half the size of ECDSAP256SHA256 public keys. Private keys of the SLH-DSA variants, on the other hand, increase in size compared to ECDSAP256SHA256 (see Table V), with the MTL mode variants storing additional data related to MTL mode.

TABLE V
SIZE OF THE ALGORITHMS' PRIVATE AND PUBLIC KEY IN BYTES.

Algorithm	Public Key	Private Key
ECDSAP256SHA256	64	32
SLH-DSA-MTL-SHA2-128s	32	211
SLH-DSA-MTL-SHAKE-128s	32	211
SLH-DSA-SHA2-128s	32	104
SLH-DSA-SHAKE-128s	32	104

D. Signature size

In MTL mode, the signature size of an RRset depends on the size of the zone, specifically the number of RRsets to sign (see Section VIII.D), the size of the binary tree that covers the RRsets (see Section V), and whether it is a condensed or full signature.

In case of the root zone with 2787 RRsets, the signature sizes are distributed as follows:

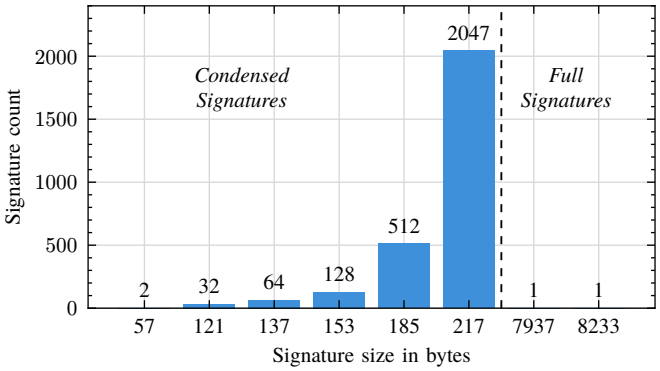

Fig. 2. Distribution of MTL mode signature sizes in the root zone.

E. Maximum signature sizes

Using the formulas mentioned earlier, we calculate the maximum signature sizes for zones sized up to 10 000 000 RRsets. As the growth is logarithmic, a variation of Figure 3 using a logarithmic scale for the x-axis can be found in Appendix B.

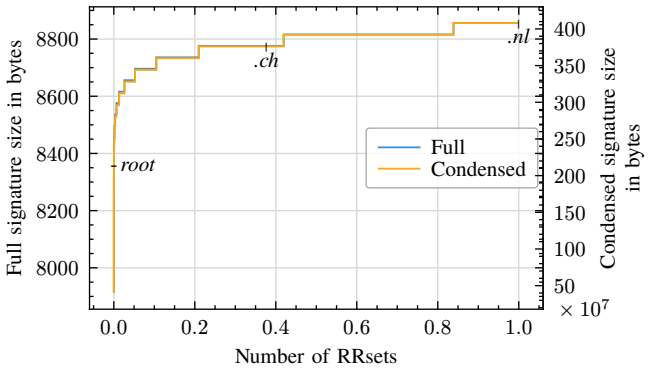

Fig. 3. Maximum signature size per number of RRsets to sign.

The root zone with 2787 RRsets has a maximum full signature size of 8376 bytes and a maximum condensed signature of 216 bytes. The `.nl` ccTLD with approximately 12 000 000 RRsets [29] has a maximum full signature size of 8856 bytes and a maximum condensed signature of 408 bytes, while the `.ch` ccTLD with 3 836 941 RRsets [30] has a maximum full signature size of 8776 bytes and a maximum condensed signature of 376 bytes.

F. DNS message sizes

As in [5], Table VI shows different DNS message sizes as reported by `dig`, where the *Delegation* column also applies to data responses that are similar in size as NS responses. Responses that fit within the 1232 bytes limit are marked in green.

Although NSEC RRs are accompanied by a condensed signature, NXDOMAIN and NODATA responses also contain the SOA RR that always carries a full signature.

TABLE VI
DNS message sizes (in bytes) when querying the root zone with 1 KSK and 1 ZSK.
(Messages smaller than or equal to 1232 bytes marked in green)

Algorithm	SOA	DNSKEY	NXDOMAIN	NODATA	Delegation
ECDSAP256SHA256	197	280	319	316	333
SLH-DSA-MTL-SHA2-128s	8366	8089	8641	8638	486
SLH-DSA-MTL-SHAKE-128s	8366	8089	8641	8638	486
SLH-DSA-SHA2-128s†	7989	8072	15903	15900	8125
SLH-DSA-SHAKE-128s†	7989	8072	15903	15900	8125

† Estimated message sizes, calculated by hand.
Queries: `. IN {NS,SOA,DNSKEY}` — *(NODATA:)* `. IN A` — *(NXDOMAIN:)* `aa. IN A`

G. Relative performance comparison

To derive the performance ratio, we divide our measured results by the results from [5]. This process is repeated for each metric (mean, variance, and file size, but not public/private key and signature size) of all performance benchmarks. The Tables VII, VIII, and IX show our previously demonstrated results alongside the results from [5] adjusted by the appropriate ratios. A list of all ratios can be found in Appendix C.

Table VII shows that from the PQC algorithms only the HAWK variants complete faster than the MTL mode algorithms in signing the root zone.

TABLE VII
Signing time in milliseconds, and file size in bytes.
Including [5] adjusted by performance ratio (lower part).

Algorithm	Mean	σ	File Size
ECDSAP256SHA256	358.17	8.64	1476949
SLH-DSA-MTL-SHA2-128s	598.52	10.03	2023946
SLH-DSA-MTL-SHAKE-128s	902.96	5.79	2023944
SLH-DSA-SHA2-128s	398793.40	776.08	30431006
SLH-DSA-SHAKE-128s	807239.86	762.97	30431006
FALCON-512	8017.19	22.7	3526586
HAWK-256	321.06	4.15	2107138
HAWK-512	428.62	8.13	3149347
SQIsign	89547.59	57.52	1762664
MAYO	1784.45	41.25	2806778
ANTRAG-512	8768.84	94.19	3274573
RSA 2048	1388.83	2.54	2130484
ECDSAP256	358.17	8.64	1476949
ED25519	395.12	5.34	1476871

While Table VIII shows that most other PQC algorithms verify the signed root zone slightly faster than the MTL mode algorithms, most complete in under one second. Only the non-MTL mode algorithms and SQIsign require more than two seconds to complete.

Key generation, as shown in Table IX, completes within a few milliseconds in all cases, although the SLH-DSA variants take significantly longer than all other algorithms.

TABLE VIII
Verification time in milliseconds.
Including [5] adjusted by performance ratio (lower part).

Algorithm	Mean	σ
ECDSAP256SHA256	550.19	4.52
SLH-DSA-MTL-SHA2-128s	598.06	8.40
SLH-DSA-MTL-SHAKE-128s	600.89	7.63
SLH-DSA-SHA2-128s	2968.52	54.94
SLH-DSA-SHAKE-128s	3476.51	22.02
FALCON-512	364.12	1.1
HAWK-256	209.7	1.41
HAWK-512	324.16	66.29
SQIsign	20148.23	35.16
MAYO	898.16	26.92
ANTRAG-512	494.81	1.41
RSA 2048	225.67	18.98
ECDSAP256	550.19	4.52
ED25519	739.06	4.52

TABLE IX
Key generation time in milliseconds.
Including [5] adjusted by performance ratio (lower part).

Algorithm	Mean	σ
ECDSAP256SHA256	2.96	0.18
SLH-DSA-MTL-SHA2-128s	22.37	0.79
SLH-DSA-MTL-SHAKE-128s	41.83	1.03
SLH-DSA-SHA2-128s	22.20	0.61
SLH-DSA-SHAKE-128s	41.85	0.68
FALCON-512	5.26	0.92
HAWK-256	3.08	0.09
HAWK-512	3.38	0.27
SQIsign	6.42	0.34
MAYO	2.96	0.23
ANTRAG-512	4.72	0.2
RSA 2048	32.4	19.86
ECDSAP256	2.96	0.18
ED25519	2.97	0.18

As for the key and signature sizes, Table X shows that most algorithms' public keys and signatures individually are smaller

than 1232 bytes, but when added together exceed 1232 bytes, which causes fragmentation in most networks.

TABLE X
SIZE OF PRIVATE/PUBLIC KEYS AND SIGNATURES IN BYTES. INCLUDING ORIGINAL RESULTS FROM [5] (LOWER PART).

Algorithm	Public Key	Private Key	Signature
ECDSAP256SHA256	64	32	64
SLH-DSA-MTL-SHA2-128s	32	211	†
SLH-DSA-MTL-SHAKE-128s	32	211	†
SLH-DSA-SHA2-128s	32	104	7856
SLH-DSA-SHAKE-128s	32	104	7856
FALCON-512	897	1281	666
HAWK-256	450	96	249
HAWK-512	1024	184	555
SQIsign	65	353	148
MAYO	1420	24	454
ANTRAG-512	768	59392	592
RSA-2048	256	1232	256
ECDSAP256	64	32	64
ED25519	32	32	64

† See Figure 3

X. DISCUSSION

In this project we measured the performance and size of two MTL mode algorithms as provided by the authors of MTL mode signatures. We verified that using MTL mode offers significant benefits in DNSSEC as outlined by the authors, reducing the signature size for all messages that do not carry the signed ladder—in the case of DNS that is data responses that are not SOA or DNSKEY. However, as the SOA RR is always part of NXDOMAIN and NODATA responses, the benefit does not apply to those responses in the current form of MTL mode signatures.

Answering research question RQ2, we verified that using MTL mode signatures greatly reduces the per-message impact of PQC algorithms with large signatures like SLH-DSA by introducing condensed signatures, while leaving the public key size unaffected.

Compared to the current DNSSEC algorithm ECDSAP256SHA256 (RQ4), we found that the MTL mode algorithms perform adequate. While increasing signature sizes significantly in all cases, the condensed signatures for responses that do not include SOA or DNSKEY RRs fit into the 1232 bytes limitation. While the public key of SLH-DSA (and by extension the MTL mode variants) is smaller than for ECDSA, its decrease in size is insignificant in the context of DNS, which can easily fit such low data volumes. Regarding the performance, the signing time almost doubles or triples, and the verification time increases by about 50 ms compared to ECDSA. Even though the performance decreases greatly for signing, it is still within reasonable limits and should not affect normal operations significantly.

Comparing to other PQC algorithms (RQ3), we found that the MTL mode algorithms perform competitive to almost all alternatives tested by O. Surý. The only algorithms faster than the MTL mode of SLH-DSA are HAWK-256 and HAWK-512 with 36–72% the time used for signing. Other algorithms require ca. 3–150 times the amount of time. However, as the performance comparison of MTL mode signatures with other PQC algorithms is based on measurements that have been adjusted by the ratio between a baseline algorithm (ECDSA) measured in this and O. Surý's research, which used different hardware and software, the accuracy of the comparison is impaired. It should still allow for an indication of the expected magnitudes of performance differences.

With regard to signature sizes, MTL mode signatures increase significantly for SOA and DNSKEY responses, while data responses benefit from the smaller condensed signatures. The public key of SLH-DSA is significantly smaller than most of the other algorithms tested by O. Surý, but this benefit is nullified by the huge DNSKEY signature. So, while MTL mode is beneficial in data responses, it negatively impacts NODATA/NXDOMAIN responses, as well as the SOA and DNSKEY RRs.

Finally, answering the main research question RQ1, the small MTL mode signatures are beneficial for most data responses, all with close performance to current signature algorithms. However, SOA and DNSKEY queries would always require transmission via TCP due to their large signatures. This also applies to NXDOMAIN and NODATA responses, as they include the SOA record. Reducing the impact on NXDOMAIN and NODATA responses would require a change of the DNS protocol, e.g. by removing the SOA RR from such responses when DNSSEC is enabled, as the NSEC RR already demonstrates non-existence.

XI. CONCLUSION

In this research we found that using MTL mode signatures in DNSSEC would have a beneficial impact on the signature size of most data responses, reducing their size compared to most PQC algorithms, albeit slightly increasing their size compared to current signature algorithms, while providing adequate signing and verification performance. However, in their current form, MTL mode signatures do not reduce the negative impact of PQC algorithms with long signatures on DNSKEY or SOA RRsets. This means that NODATA and NXDOMAIN responses, as they contain the SOA RR, retain the long signatures of a chosen PQC algorithm. Changes to the DNS protocol could enable the benefits of MTL mode signatures for NODATA and NXDOMAIN responses (see Section XII).

While the chosen methodology for comparison with other PQC algorithms limits the accuracy of the comparison results presented earlier, the general comparison can still inform about the magnitude of performance differences between said algorithms and MTL mode signatures.

Going forward, improvements to the Domain Name System and the Internet-Draft on MTL mode signatures in DNSSEC should be investigated as outlined in Section XII.

XII. Future Work

During this project we discovered a few opportunities to improve the usage of MTL mode signatures in DNSSEC, as well as some avenues for further research.

A. EDNS(0) option: stored ladder version

One option to reduce the impact of MTL mode signatures on SOA and DNSKEY queries would be to investigate an EDNS(0) option to indicate to the authoritative name server which ladder is currently available to/stored by the client. This way the name server could send a condensed signature for all SOA (and DNSKEY) queries, if the ladder stored by the resolver is up-to-date. Additionally, the authoritative could keep track of old ladders and check whether the response in question is covered by the old ladder. If the resolver's ladder does not authenticate the current SOA record, the authoritative name server can send a truncation response and deliver the full signature when queried by the resolver via TCP. The interplay between this option and the option to actively request a full signature from the current Internet-Draft version would need to be investigated.

B. SOA in DNSSEC enabled non-existence proofs

To reduce the impact on denial of existence proofs, there are two options. One, always send a condensed signature for SOA records that are part of NSEC proofs, or when the EDNS(0) option from the previous section is used. Two, remove the SOA record from NODATA and NXDOMAIN responses when DNSSEC is enabled and NSEC/NSEC3 records would be returned. DNSSEC unaware resolvers would not set the DNSSEC OK flag, which would signal to the authoritative to use the SOA record for denial of existence responses.

C. Impact on dynamic zones

Signing new RRsets requires extending or replacing the Merkle tree ladder. In the current version of the draft with the binary rung strategy, frequent zone changes would require frequent transmissions of fresh ladders. Future projects could measure the impact on traffic and frequency of outdated ladders for dynamic zones.

Alternatively, as mentioned previously (in Section XII.A), multiple ladder versions could be kept around and re-used if applicable to the queries. And new trees could be added to the ladder for some time instead of replacing the smaller trees.

D. Repeat benchmarks of all algorithms

For a more accurate comparison, the benchmarks of all or an updated selection of PQC algorithms could be repeated. This could then help reinforce a future decision for selecting one or more PQC algorithms for use in DNSSEC.

E. Measure performance of fast variants of SLH-DSA-XXX-128

Even though the faster 128bit variants of SLH-DSA SLH-DSA-SHA2-128f and SLH-DSA-SHAKE-128f have even larger signatures (about 17 kB) they could still be a viable alternative, if their performance is significantly greater than the short 128s variants, as the full signatures require TCP even with the short variant.

Acknowledgements

Thanks to my supervisor and my colleagues for their support in this research.

References

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," Nov. 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700.

[2] F. K. Wilhelm *et al.*, "Status of quantum computer development," Jan. 2025. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/Entwicklungstand_QC_V_2_1.pdf?__blob=publicationFile&v=3

[3] Computer Security Division, Information Technology Laboratory, "Post-Quantum Cryptography." Accessed: Jun. 03, 2025. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[4] Computer Security Division, Information Technology Laboratory, "Selected Algorithms - Post-Quantum Cryptography." Accessed: Jun. 04, 2025. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms

[5] O. Surý, "PQC FOR DNSSEC." [Online]. Available: https://ripe90.ripe.net/presentations/40-PQC-FOR-DNSSEC.pdf

[6] 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 Comput. Commun. Rev.*, vol. 50, no. 4, pp. 49–57, Oct. 2020, doi: 10.1145/3431832.3431838.

[7] A. Koolhaas and T. Slokkker, "Defragmenting DNS," Jul. 2020, [Online]. Available: https://nlnetlabs.nl/downloads/publications/os3-2020-rp2-defragmenting-dns.pdf

[8] J. Harvey, B. Kaliski, A. Fregly, and S. Sheth, "Merkle Tree Ladder (MTL) Mode Signatures," Mar. 2025. Accessed: Jun. 03, 2025. [Online]. Available: https://datatracker.ietf.org/doc/draft-harvey-cfrg-mtl-mode-06

[9] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ Signature Framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, in CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2129–2146. doi: 10.1145/3319535.3363229.

[10] C. Schutijser, R. Koning, and E. Lastdrager, "Post-quantum Algorithm Testing and Analysis for the DNS." Accessed: Jun. 04, 2025. [Online]. Available: https://www.sidnlabs.nl/en/news-and-blogs/a-quantum-safe-cryptography-dnssec-testbed

[11] J. Goertzen, P. Thomassen, and N. Wisiol, "Field Experiments on Post-Quantum DNSSEC." [Online]. Available: https://datatracker.ietf.org/meeting/120/materials/slides-120-maprg-field-experiments-on-post-quantum-dnssec-01.pdf

[12] RIPE NCC, "RIPE Atlas." Accessed: Jun. 04, 2025. [Online]. Available: https://atlas.ripe.net/

[13] ISC, "BIND ondrej/pqc-main." Accessed: Jun. 04, 2025. [Online]. Available: https://gitlab.isc.org/isc-projects/bind9/-/tree/ondrej/pqc-main

[14] A. Fregly, J. Harvey, B. Kaliski, and D. Wessels, "Stateless Hash-Based Signatures in Merkle Tree Ladder Mode (SLH-DSA-MTL) for DNSSEC," Apr. 2025. Accessed: Jun. 03, 2025. [Online]. Available: https://datatracker.ietf.org/doc/draft-fregly-dnsop-slh-dsa-mtl-dnssec-04

[15] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard," Aug. 2024. doi: 10.6028/NIST.FIPS.204.

[16] National Institute of Standards and Technology, "Stateless Hash-Based Digital Signature Standard," Washington, D.C., Aug. 2024. doi: 10.6028/nist.fips.205.

[17] "DNS flag day 2020." Accessed: Jun. 30, 2025. [Online]. Available: https://www.dnsflagday.net/2020/

[18] 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," 2023. [Online]. Available: https://eprint.iacr.org/2022/1730.pdf

[19] National Institute of Standards and Technology, "Digital Signature Standard (DSS)," Feb. 2023. doi: 10.6028/NIST.FIPS.186-5.

[20] P. Wouters and O. Surý, "Algorithm Implementation Requirements and Usage Guidance for DNSSEC," Jun. 2019. doi: 10.17487/RFC8624.
[21] J. Harvey and S. Sheth, "verisign/mtl-mode-ldns." Accessed: Jun. 03, 2025. [Online]. Available: https://github.com/verisign/mtl-mode-ldns
[22] The kernel development community, "The kernel's command-line parameters." 2025. Accessed: Jun. 09, 2025. [Online]. Available: https://www.kernel.org/doc/html/v4.14/admin-guide/kernel-parameters.html
[23] R. M. Love, "taskset - set or retrieve a process's CPU affinity." Mar. 29, 2025.
[24] D. Peter, "hyperfine." Accessed: Jun. 10, 2025. [Online]. Available: https://github.com/sharkdp/hyperfine
[25] NLnet Labs, "NLnetLabs/NSD." Accessed: Jun. 03, 2025. [Online]. Available: https://github.com/NLnetLabs/nsd/tree/ietf121-hackathon/slh-dsa-mtl-dnssec
[26] J. Harvey and S. Sheth, "verisign/mtl-mode-unbound." Accessed: Jun. 03, 2025. [Online]. Available: https://github.com/verisign/mtl-mode-unbound
[27] "Root Zone File." [Online]. Available: https://www.iana.org/domains/root/files
[28] D. Wessels, P. Barber, M. Weinberg, W. Kumari, and W. Hardaker, "Message Digest for DNS Zones," Feb. 2021. doi: 10.17487/RFC8976.
[29] "List of .nl Zone Domains." [Online]. Available: https://netapi.com/nl-domains/
[30] ".CH Zone File." [Online]. Available: https://portal.switch.ch/pub/opendata/#tab-fccd70a3-b98e-11ed-9a74-5254009dc73c-3

Appendix A
List of Abbreviations

DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDSAP256SHA256	ECDSA Curve P-256 with SHA-256
EDNS(0)	Extension Mechanisms for DNS
KSK	Key Signing Key
MTL	Merkle tree ladder
NIST	National Institute of Standards and Technology
PQC	Post-Quantum Cryptography
RR	Resource Record
RRset	Resource Record set
SLH-DSA	Stateless Hash-Based Digital Signature Algorithm
SLH-DSA-MTL	Stateless Hash-Based Digital Signature Algorithm in Merkle Tree Ladder mode
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ZSK	Zone Signing Key
ccTLD	country code top-level domain

Appendix B
Maximum signature sizes plot

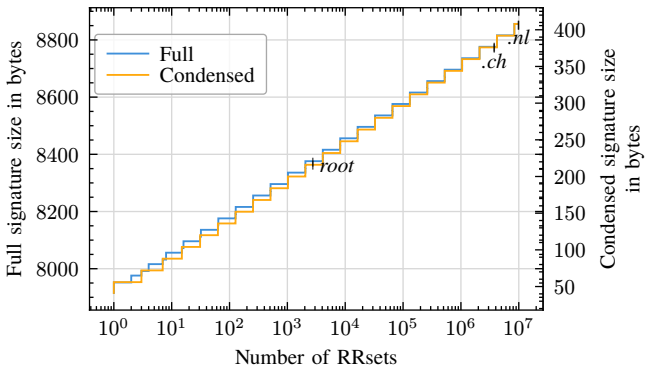

Fig. 4. Maximum signature size per number of RRsets to sign.

Appendix C
Collection of performance ratios

The following table lists the performance ratios based on the ECDSAP256SHA256 results from this research and from [5] for each metric:

TABLE XI
Performance ratios rounded to five decimals.

Metric	Ratio
Signing time	1.64223
Signing time variance	0.84706
Signed file size	1.21955
Verification time	0.90195
Verification time variance	1.00444
Key generation time	0.06563
Key generation time variance	0.07826