DNSEXT
Internet-Draft
Intended status: Experimental
Expires: January 5, 2013

R. Gieben
SIDN Labs
W. Mekking
NLnet Labs

July 04, 2012

DNS Security (DNSSEC) Authenticated Denial of Existence
draft-gieben-nsec4-01

Abstract

The Domain Name System Security (DNSSEC) Extensions introduced the NSEC resource record for authenticated denial of existence, and the NSEC3 resource record for hashed authenticated denial of existence. This document introduces an alternative resource record, NSEC4, which similarly provides authenticated denial of existence. It permits gradual expansion of delegation-centric zones, just like NSEC3 does. With NSEC4 it is possible, but not required, to provide measures against zone enumeration.

NSEC4 reduces the size of the denial of existence response and adds Opt-Out to unhashed names.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2013.

Table of Contents

1. Introduction

1.1. Rationale

Hashed authenticated denial of existence proofs frequently hinge on the closest encloser proof (Section 7.2.1 and 8.3 of [RFC5155]). When validating a hashed denial of existence response, a validator must deny or assert the presence of a next closer name and a wildcard name. A validator can derive these names from the closest encloser.

This is why most of the denial of existence responses with NSEC3 contain three records:

1. A record which matches the closest encloser, this tells the validator what the (unhashed) name of the closest encloser is;

2. A record which covers or matches the next closer, to deny or assert the existence of the next closer name. The validator needs to know the closest encloser to construct the next closer name;

3. A record which covers or matches the wildcard, to deny or assert wildcard synthesis. The validator needs to know the closest encloser to construct the source of synthesis.

This document presents a new record, NSEC4, that is similar to NSEC3, but differs in the following ways:

- It provides a new way to deny the existence of the wildcard, by introducing the Wildcard flag (described in Section 3.1.2.2). This bit makes the third record, from the list above, redundant;

- It allows for unhashed records, by introducing an Identity function (described in Section 3.1.1).

With NSEC4 you will need a maximum of two records for any denial of existence response, saving one record and accompanying signature(s) compared to NSEC3.

By defining an Identity function, we also fold back NSEC into NSEC4 and add Opt-out to unhashed names. With this change we collapse NSEC and NSEC3 into one new record to leave only one form of authenticated denial of existence in the DNS.

1.2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this

document are to be interpreted as described in [RFC2119].

1.3. Terminology

The reader is assumed to be familiar with the basic DNS and DNSSEC concepts described in [RFC1034], [RFC1035], [RFC4033], [RFC4034], [RFC4035], and subsequent RFCs that update them: [RFC2136], [RFC2181], [RFC2308] and [RFC5155].

Furthermore, the same terminology is used throughout this document as is described in Section 1.3 from [RFC5155], with the following changes:

Original owner name: the owner name corresponding to a hashed owner name if hashing is used. Or the owner name as-is if no hashing is used.

Opt-Out NSEC4 RR: an NSEC4 RR that has the Opt-Out flag set to 1.

Wildcard NSEC4 RR: an NSEC4 RR that has the Wildcard flag set to 1.

Opt-Out zone: a zone with at least one Opt-Out NSEC4 RR.

Base32: the "Base 32 Encoding with Extended Hex Alphabet" as specified in [RFC4648]. Note that trailing padding characters ("=") are not used in the NSEC4 specification.

To cover: an NSEC4 RR is said to "cover" a name if the (hashed) name or (hashed) next closer name falls between the owner name of the NSEC4 RR and the next (hashed) owner name of the NSEC4. In other words, if it proves the nonexistence of the name, either directly or by proving the nonexistence of an ancestor of the name.

To match: When a hash algorithm is defined, an NSEC4 RR is said to "match" a name if the owner name of the NSEC4 RR is the same as the hashed owner name of that name. When no hash algorithm (Identity function) is defined, an NSEC4 RR is said to "match" a name if the name and the owner name of the NSEC4 RR are equal.

Identity function: Perform no hashing. Leave the name as-is.

2. Experimental Status

This document describes an EXPERIMENTAL extension to DNSSEC. It interoperates with non-experimental DNSSEC using the technique described in [RFC4955]. This experiment is identified with the following private algorithm (using algorithm PRIVATEDNS):

- Algorithm "5.nsec4.nlnetlabs.nl.", is an alias for algorithm 5, RSASHA1.

Servers wishing to sign and serve zones that utilize NSEC4 MUST sign the zone with this private algorithm and MUST NOT use any other algorithms.

Resolvers MUST NOT apply NSEC4 validation described in this document unless a response contains RRSIGs created with this private algorithm.

3. The NSEC4 Resource Record

The NSEC4 RR provides authenticated denial of existence for DNS RRsets.

The NSEC4 RR lists RR types present at the original owner name of the NSEC4 RR. It includes the next (hashed) owner name in the canonical order of the zone. The complete set of NSEC4 RRs in a zone indicates which RRSets exist for the original owner name of the RR and form a chain. This information is used to provide authenticated denial of existence for DNS data. To provide protection against zone enumeration, the owner names used in the NSEC4 RR can be cryptographic hashes of the original owner name prepended as a single label to the name of the zone. The NSEC4 RR indicates which hash function (if any) is used to construct the hash, which salt is used, and how many iterations of the hash function are performed over the original owner name.

The hashing technique is the same as with NSEC3 and is described in Section 5 of [RFC5155]. NSEC3 creates hashes for empty non-terminals, NSEC4 does the same, even when the Identity function is in use.

(Hashed) owner names of unsigned delegations may be excluded from the chain. An NSEC4 RR whose span covers an owner name or next closer name of an unsigned delegation is referred to as an Opt-Out NSEC4 RR and is indicated by the presence of a flag.

If hashing is in use, the owner name for the NSEC4 RR is the base32 encoding of the hashed owner name prepended as a single label to the name of the zone.

The type value for the NSEC4 RR is [TBD].

The NSEC4 RR RDATA format is class independent and is described below.

The class MUST be the same as the class of the original owner name.

The NSEC4 RR SHOULD have the same TTL value as the SOA minimum TTL field. This is in the spirit of negative caching [RFC2136].

3.1. RDATA Fields

The NSEC4 RDATA has many similarities with the NSEC3 RDATA, but there are differences:

- There is an extra flag bit reserved to indicate whether wildcard synthesis is possible (e.g. does a wildcard domain name exist that is immediately descending from the original owner name?);

- The hash length does not need to be stored, as all domain names are stored as domain names, not raw hashes.

[MM: Hash length is one byte. In general, NSEC3 is used in TLD zones. Those domain names are relatively short (on average 3 characters (5 bytes wireformat), so in that case NSEC3 RRs become 4 bytes longer.]

3.1.1. Hash Algorithm

[RFC5155] defines the NSEC3 hash algorithm registry. Hash algorithm 0 is reserved. For NSEC4 we define hash algorithm 0 to be the Identity function, meaning that no hashing is used.

3.1.2. Flags

The Flags field is identical to the Flags field as defined in [RFC5155]. This specification adds a new flag, the Wildcard Flag.

3.1.2.1. Opt-Out Flag

Like the Opt-Out Flag defined in Section 3.1.2.1 of [RFC5155].

3.1.2.2. Wildcard Flag

The Wildcard Flag indicates whether there is wildcard synthesis possible (e.g. does a wildcard domain name exist that is immediately descending from the original owner name of the NSEC4?).

If the Wildcard flag is set, wildcard synthesis is possible.

If the Wildcard flag is clear, wildcard synthesis is not possible.

3.1.3. Iterations

Like the Iterations field defined in Section 3.1.3 of [RFC5155].

3.1.4. Salt Length

Like the Salt Length field defined in Section 3.1.4 of [RFC5155].

3.1.5. Salt

Like the Salt field defined in Section 3.1.5 of [RFC5155].

3.1.6. Next (Hashed) Owner Name

The Next Owner Name field contains the next owner name that exists in the definition of Section 2.2.3 of [RFC4592].

The field contains the next owner name in the canonical ordering of the zone, as explained in Section 6.1 of [RFC4034].

A sender MUST NOT use DNS name compression on the Next Owner Name field when transmitting an NSEC4 RR.

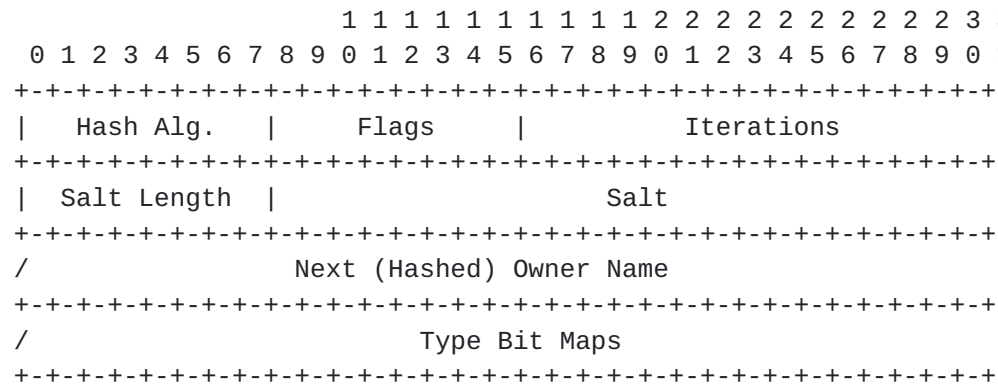
Owner names of RRsets for which the given zone is not authoritative (such as glue records) MUST NOT be listed in the Next Owner Name, unless at least one authoritative RRset exists at the same owner name.

3.1.7. Type Bit Maps

Like the Type Bit Maps field defined in Section 3.1.8 of [RFC5155].

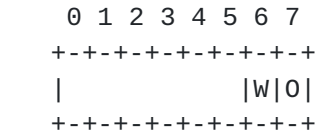
3.2. NSEC4 RDATA Wire Format

The RDATA of the NSEC4 RR is as shown below.



Hash Algorithm is a single octet. If Hash Algorithm is zero (Identity function), the Iterations field, the Salt Length field and the Salt field MUST be ignored.

Flags field is a single octet. The following one-bit flags are defined:



o O - Opt-Out flag

o W - Wildcard flag

Iterations is represented as a 16-bit unsigned integer, with the most significant bit first.

Salt Length is represented as an unsigned octet. Salt Length represents the length of the Salt field in octets. If the value is zero, the following Salt field is omitted.

Salt, if present, is encoded as a sequence of binary octets. The length of this field is determined by the preceding Salt Length field.

If Hash Algorithm is not zero, the Next (Hashed) Owner Name is a base32 encoded domain name of the hashed next owner name prepended as a single label to the name of the zone. If Hash Algorithm is zero it is a plain domain name.

The Type Bit Maps encode the existing types at the original owner name that matches the NSEC4 RR.

3.2.1. Type Bit Maps Encoding

The encoding of the Type Bit Maps field is the same as that used by the NSEC and NSEC3 RR, described in [RFC4034], as well as in [RFC5155].

3.3. Presentation Format

The presentation format of the RDATA portion is as follows:

o The Hash Algorithm field is represented as an unsigned decimal integer. The value has a maximum of 255.

- The Flags field is represented as an unsigned decimal integer. The value has a maximum of 255.

- The Iterations field is represented as an unsigned decimal integer. The value is between 0 and 65535, inclusive.

- The Salt Length field is not represented.

- The Salt field is represented as a sequence of case-insensitive hexadecimal digits. Whitespace is not allowed within the sequence. The Salt field is represented as "-" (without the quotes) when the Salt Length field has a value of 0.

- The Next (Hashed) Owner Name field is represented as a domain name.

- The Type Bit Maps field is represented as a sequence of RR type mnemonics. When the mnemonic is not known, the TYPE representation as described in Section 5 of [RFC3597] MUST be used.

3.3.1. Examples

NSEC record:

```
example. NSEC a.example NS SOA RRSIG DNSKEY NSEC
```

The same data shown as an NSEC3 record:

```
3msev9usmd4br9s97v51r2tdvmr9iqo1.example. NSEC3 1 0 0 - (
                    6cd522290vma0nr8lqu1ivtcofj94rga
                    NS SOA RRSIG DNSKEY NSEC3PARAM )
```

As an NSEC4 record with Identity function:

```
example. NSEC4 0 0 0 - a.example. NS SOA RRSIG DNSKEY NSEC4 NSEC4PARAM
```

And as an NSEC4 record with SHA1 hashing:

```
3msev9usmd4br9s97v51r2tdvmr9iqo1.example. NSEC4 1 0 0 - (
             6cd522290vma0nr8lqu1ivtcofj94rga.example.
             NS SOA RRSIG DNSKEY NSEC4PARAM )
```

4. The NSEC4PARAM Resource Record

Exactly like NSEC3PARAM described in Section 5 of [RFC5155], except the type code used [TBD] is that of NSEC4PARAM.

5. Opt-Out

This specification adds Opt-Out as described in Section 6 of [RFC5155]. Because of the Identity function, this allows for Opt-Out being used with unhashed names. A similar method is described in [RFC4956], but with NSEC4 we can reuse the Opt-Out bit from the Flags field.

6. Empty non-terminals

With NSEC3, every empty non-terminal will have a NSEC3 record. This is mentioned in [RFC5155], for instance in section 7.1, the second bullet point:

> Each empty non-terminal MUST have a corresponding NSEC3 RR, unless the empty non-terminal is only derived from an insecure delegation covered by an Opt-Out NSEC3 RR.

This is a crucial difference with respect to NSEC, where no such provision exists.

With NSEC4 we unify NSEC and NSEC3 and consequently, each empty non-terminal will get an NSEC4 record (see Section 7.1, the 6th bullet). Furthermore, NSEC4 represents the next owner name as a domain name, like NSEC, while NSEC3 represents the next name as an unmodified binary hash value.

Due to these changes, we can revert back to canonical ordering for NSEC4. This greatly simplifies the comparison code, because there is only one ordering mechanism.

7. Authoritative Server Considerations

7.1. Zone Signing

Zones using NSEC4 must satisfy the same properties as described in Section 7.1 of [RFC5155], with NSEC3 replaced by NSEC4.

In addition, for each original owner name that has a wildcard domain name immediately descending from the original owner name, the corresponding NSEC4 RR MUST have the Wildcard bit set in the Flags field.

The following steps describe one possible method of proper construction of NSEC4 RRs.

1. Select the hash algorithm and the values for salt and iterations;

2. For each unique original owner name in the zone add an NSEC4 RR;

   * If Opt-Out is being used, owner names of unsigned delegations MAY be excluded;

   * The owner name of the NSEC4 RR is either the hash of the original owner name, prepended as a single label to the zone name, or is equal to the original owner name if Identity function is used;

   * The Next Owner Name field is left blank for the moment;

   * If Opt-Out is being used, set the Opt-Out bit to one.

3. For collision detection purposes, if hashing is used, optionally keep track of the original owner name with the NSEC4 RR. Create an additional NSEC4 RR corresponding to the original owner name with the asterisk label prepended. Mark this NSEC4 RR as temporary;

4. If the original owner name is a wildcard domain name (Section 2.1.1. of [RFC4592]), mark the NSEC4 to be an NSEC4 RR that is matching a wildcard;

5. For each RRSet at the original owner name, set the corresponding bit in the Type Bit Maps field;

6. Additional NSEC4 RRs need to be added for every empty non-terminal between the apex and the original owner name. If hashing is used, optionally keep track of the original owner names of these NSEC4 RRs and create temporary NSEC4 RRs for wildcard collisions in a similar fashion to step 3;

7. Sort the set of NSEC4 RRs into canonical order.

8. Combine NSEC4 RRs with identical owner names by replacing them with a single NSEC4 RR with the Type Bit Maps field consisting of the union of the types represented by the set of NSEC4 RRs. If hashing is used and the original owner name was tracked, then collisions may be detected when combining, as all of the matching NSEC4 RRs should have the same original owner name. If a hash collision is detected, then a new salt has to be chosen, and the signing process is restarted. Discard any possible temporary NSEC4 RRs;

9. In each NSEC4 RR, insert the next (hashed) owner name by using the domain name of the next NSEC4 RR in canonical order. The next (hashed) owner name of the last NSEC4 RR in the zone contains the value of the (hashed) owner name of the first NSEC4 RR in canonical order.

   If the NSEC4 is marked to be matching a wildcard, find the NSEC4 that matches the closest encloser. Set the Wildcard bit in the Flags field of that NSEC4;

10. Finally, add an NSEC4PARAM RR with the same Hash Algorithm, Iterations, and Salt fields to the zone apex.

7.2. Zone Serving

This specification modifies DNSSEC-enabled DNS responses generated by authoritative servers. In particular, it replaces the use of NSEC or NSEC3 RRs in such responses with NSEC4 RRs.

7.2.1. Denial of Wildcard Synthesis Proof

Instead of wasting a whole denial of existence RR to deny a wildcard, we have introduced a bit in the Flags field of the NSEC4 RR that indicates whether wildcard synthesis was possible because there exists a wildcard domain name immediately descending from the original owner name.

The Denial of Wildcard Synthesis proof consists of one NSEC4 RR, that matches some domain name, and that has the Wildcard bit clear.

Note that without much knowledge of the original owner name, this proof is not really useful. In particular, we don't know if this is the wildcard synthesis that we are looking for. This changes if we combine this proof with the closest encloser proof.

7.2.2. Closest Encloser Proof

For some NSEC4 responses, namely Name Error Response (Section 7.2.4) and Referrals to Unsigned Subzones (Section 7.2.8), a proof of the closest encloser is required. This is a proof that some ancestor of the QNAME is the closest encloser of QNAME. The proof is described in Section 7.2.1 of [RFC5155], and is the same for NSEC4.

7.2.3. Denial of Source of Synthesis Proof

The denial of wildcard synthesis proof combined with the closest encloser proof results in a denial of source of synthesis proof. The source of synthesis is defined in [RFC4592] as the wildcard domain

name immediately descending from the closest encloser.

The Denial of Source of Synthesis proof consists of (up to) two NSEC4 RRs, the same that constructed the closest encloser proof:

- an NSEC4 RR that matches the closest encloser, and that has the Wildcard bit clear in the Flags field;
- an NSEC4 RR that covers the next closer name to the closest encloser.

The first NSEC4 RR essentially proves that the encloser exists, and that no wildcard synthesis at the encloser is possible. The second NSEC4 RR proves that the encloser is the closest, thus the denial of the wildcard synthesis is the denial of the source of synthesis.

7.2.4. Name Error Responses

If the zone does not contain any RRsets matching QNAME either exactly or via wildcard name expansion, then the name server must include proof that:

- there is no exact match for QNAME;
- the zone contains no RRsets that would match QNAME via wildcard name expansion.

With NSEC, the server includes in the response an NSEC RR that covers QNAME, and an NSEC RR that covers the wildcard RR at the closest encloser.

With NSEC3, the server includes in the response an NSEC3 RR that covers the next closer, an NSEC3 RR that covers the wildcard RR at the closest encloser, and an NSEC3 RR that matches the closest encloser.

To prove the nonexistence of QNAME with NSEC4, the server MUST include a denial of source of synthesis proof. This collection of (up to) two NSEC4 RRs proves both that QNAME does not exist and that a wildcard that could have matched QNAME also does not exist.

7.2.5. No Data Responses

7.2.5.1. QTYPE is not DS

When a NODATA response needs to be returned, it is safe to say that QNAME exists. Similar to NSEC and NSEC3, server MUST include the NSEC4 RR that matches QNAME. This NSEC4 RR MUST NOT have the bits

corresponding to either the QTYPE or CNAME set in its Type Bit Maps field.

7.2.5.2. QTYPE is DS

Because of Opt-Out, the response can be different when QTYPE is DS. If no NSEC4 RR matches QNAME, the server MUST return a closest provable encloser proof for QNAME. The NSEC4 RR that covers the next closer name MUST have the Opt-Out bit set.

Note that we do not need to ensure the denial of source of synthesis proof, because a DS RRset next to a wildcard is meaningless (Section 4.6, [RFC4592]).

7.2.6. Wildcard Answer Responses

If the zone does not contain any RRsets matching QNAME, but there is wildcard name expansion possible then the name server must include proof that the wildcard match was valid. This proof is accomplished by proving that QNAME does not exist and that the closest encloser of QNAME and the immediate ancestor of the wildcard are equal.

Both with NSEC and NSEC3, the server includes in the response an NSEC RR that covers the next closer. It is not necessary to return a RR that matches the closest encloser, as the existence of this closest encloser is proven by the presence of the expanded wildcard in the response.

To prove that the wildcard name expansion was valid with NSEC4, the server MUST include in the response an NSEC4 RR that covers the next closer. For the same reasons as with NSEC and NSEC3, it is not necessary to return a RR that matches the closest encloser.

7.2.7. Wildcard No Data Responses

With NSEC, the server includes in the response an NSEC RR that matches the wildcard, in addition to the NSEC RR that covers the next closer. The NSEC RR does not have the bits corresponding to QTYPE or CNAME set in its Type Bit Maps field.

Again, with NSEC3, the server includes in the response an NSEC3 RR that matches the wildcard, in addition to the NSEC3 RR that covers the next closer. The NSEC3 RR does not have the bits corresponding to QTYPE or CNAME set in its Type Bit Maps field. Besides that, an NSEC3 RR that matches the closest encloser is included, because there was no expanded wildcard in the response that can be used to determine the closest encloser.

[RFC5155] already notes that the closest encloser to QNAME must be the immediate ancestor of the wildcard RR, which is also defined in [RFC4592]. A closest encloser proof is not necessitated.

To prove the wildcard no data response with NSEC4, the server MUST include in the response an NSEC4 RR that matches the wildcard, and an NSEC4 RR that covers the next closer. The closest encloser can be derived from the NSEC4 RR that matches the wildcard. From that, the next closer can be derived.

7.2.8. Referrals to Unsigned Subzones

If there is an NSEC4 RR that matches the delegation name, then that NSEC4 RR MUST be included in the response. The DS and CNAME bit in the type bit maps of the NSEC4 RR must not be set (by definition).

If the zone is Opt-Out, then there may not be an NSEC4 RR corresponding to the delegation. In this case, the closest provable encloser proof MUST be included in the response. The included NSEC4 RR that covers the next closer name for the delegation MUST have the Opt-Out flag set to one.

Note that with the Identity function, the NSEC4 RR that matches the closest provable encloser does not need to be included in the response, as it can be derived from the NSEC4 that covers the next closer name.

7.2.9. Responding to Queries for NSEC4 Only Owner Names

When NSEC4 hashing is in effect the paradox (NSEC4 records deny their own existence) described in Section 7.2.8 of [RFC5155] is back. When the Identity function is used, there is no paradox. In light of this, queries for the NSEC4 resource type are handled in the same way as normal queries. Resolvers initiating these queries SHOULD disregard any information learned from the returned NSEC4 records.

7.2.10. Server Response to a Run-Time Collision

The same considerations as described in Section 7.2.9 of [RFC5155] for NSEC3 apply to NSEC4.

7.3. Secondary Servers

The same considerations as described in Section 7.3 of [RFC5155] for NSEC3 and NSEC3PARAM apply to NSEC4 and NSEC4PARAM.

7.4. Zones Using Unknown Hash Algorithms

The same considerations as described in Section 7.4 of [RFC5155] for NSEC3 apply to NSEC4.

7.5. Dynamic Update

A zone signed using NSEC4 may accept dynamic updates [RFC2136]. However, NSEC4 introduces some special considerations for dynamic updates.

Adding and removing names in a zone MUST account for the creation or removal of empty non-terminals, similar to [RFC5155], Section 7.5.

The presence of Opt-Out in a zone means that some additions or removals of unsigned delegations of names will not require changes to the NSEC4 RRs in a zone. The same considerations as in [RFC5155], Section 7.5 for NSEC3 apply for NSEC4.

The presence of Opt-Out in a zone means that when adding or removing NSEC4 RRs, the value of the Opt-Out flag that should be set in new or modified NSEC4 RRs is ambiguous. Servers SHOULD follow the set of basic rules to resolve the ambiguity, as described in [RFC5155], Section 7.5.

Adding and removing wildcard names in a zone MUST account for the setting or clearing of the Wildcard bit in the Flags field:

- When adding a wildcard name, the NSEC4 RR that matches the immediate parent of the wildcard MUST set the Wildcard bit in the Flags field;

- When deleting a wildcard name, the NSEC4 RR that matches the immediate parent of the wildcard MUST clear the Wildcard bit in the Flags field.

8. Validator Considerations

8.1. Responses with Unknown Hash Types

A validator MUST ignore NSEC4 RRs with unknown hash types. The practical result of this is that responses containing only such NSEC4 RRs will generally be considered bogus.

8.2. Verifying NSEC4 RRs

A validator MUST ignore the undefined bits (0-5) in the Flags field of NSEC4 RRs.

A validator MAY treat a response as bogus if the response contains NSEC4 RRs that contain different values for hash algorithm, iterations, or salt from each other for that zone.

8.3. Validating Name Error Responses

A validator MUST verify that there is a closest encloser for QNAME present in the response. A validator MUST verify that the Wildcard bit is clear in the Flags field of the NSEC4 RR that matches the closest encloser.

> Note: In denial of existence responses, the Wildcard flag will never be set. Setting the bit indicated that wildcard synthesis is possible at the closest encloser. Obviously, that contradicts with the denial of existence of the query name. Nevertheless, a validator must verify that the Wildcard bit is clear. If a validator fails to check the bit, it is vulnerable to replay attacks. For example, if you do not check the Wildcard Flag in the example.com NSEC4 (but *.example.com does exist), an attacker can use the record to deny names that would otherwise match the wildcard name.

In order to find the closest encloser, the validator MUST find the longest name, X, such that X is an ancestor of QNAME that is matched by an NSEC4 RR present in the response.

One possible algorithm for finding the closest encloser is as follows:

1. Set SNAME=QNAME;

2. If there is an NSEC4 RR in the response that matches SNAME, then we have found the closest encloser;

3. Truncate SNAME by one label from the left, go to step 2.

Once the closest encloser has been discovered, the validator MUST check that the NSEC4 RR that has the closest encloser as the original owner name is from the proper zone. The DNAME type bit MUST NOT be set and the NS type bit MUST be clear if the SOA type bit is clear.

If this is not the case, it would be an indication that an attacker is using them to falsely deny the existence of RRs for which the server is not authoritative.

In addition, the validator MUST verify that there is an NSEC4 RR that covers the next closer name.

8.4. Validating No Data Responses

If QTYPE is not DS, a validator MUST verify that an NSEC4 RR that matches QNAME is present and that both the QTYPE and the CNAME type are not set in its Type Bit Maps field.

Note that this test also covers the case where the NSEC4 RR exists because it corresponds to an empty non-terminal, in which case the NSEC4 RR will have an empty Type Bit Maps field.

If QTYPE is DS, and there is an NSEC4 RR that matches QNAME present in the response, then that NSEC4 RR MUST NOT have the bits corresponding to DS and CNAME set in its Type Bit Maps field.

If there is no such NSEC4 RR, then the validator MUST verify that there is a closest provable encloser for QNAME present in the response. The closest provable encloser is found in a similar way as the closest encloser. In addition, the validator MUST verify that there is an NSEC4 RR that covers the next closer name and has the Opt-Out bit set.

8.5. Validating Wildcard Answer Responses

The verified wildcard answer RRSet in the response provides the validator with a closest encloser for QNAME. The validator can do so by checking the label count in the RRSIG and the number of labels in the answer's owner name.

The validator MUST verify that there is an NSEC4 RR that covers the next closer name to QNAME is present in the response. This proves that QNAME itself did not exist and that the correct wildcard was used to generate the response.

8.6. Validating Wildcard No Data Responses

The validator MUST verify that there is an NSEC4 RR present in the response that matches the source of synthesis.

In order to find the source of synthesis, the validator MUST find the longest name, X, such that X is an ancestor of QNAME and that *.X is matched by a NSEC4 RR present in the response.

One possible algorithm for finding the source of synthesis is as follows:

1. Set SNAME=QNAME;

2. Truncate SNAME by one label from the left. This is a candidate for the closest encloser;

3. Set WNAME to be SNAME with the asterisk label prepended: WNAME=*.SNAME;

4. If there is an NSEC4 RR in the response that matches WNAME, then we have found the source of synthesis, with SNAME being the closest encloser;

5. Go to step 2.

The validator does not need to check that the closest encloser is from the proper zone. The authoritative server returned an NSEC4 that matches the source of synthesis. According to [RFC6672], this proves that the server did not encounter a referral (step 3b of the server algorithm [RFC1035]), nor did it encounter a DNAME (step 3c of the server algorithm [RFC1035]).

Now that the validator knows the source of synthesis and thus the closest encloser, it can derive the next closer name. The validator MUST verify that there is an NSEC4 RR that covers the next closer name to QNAME, is present in the response.

Note that, because the response included an NSEC4 that matches the source of synthesis, we know that there exists data in the zone below the closest encloser. Therefore, the closest encloser cannot be a delegation, nor can there exists a DNAME RRset at the closest encloser.

[MM: As an additional check, the validator can verify if the NSEC4 matching the closest encloser has the Wildcard Flag set.]

8.7. Validating Referrals to Unsigned Subzones

The delegation name in a referral is the owner name of the NS RRSet present in the authority section of the referral response.

If there is an NSEC4 RR present in the response that matches the delegation name, then the validator MUST ensure that the NS bit is set and that the DS bit is not set in the Type Bit Maps field of the NSEC4 RR. The validator MUST also ensure that the NSEC4 RR is from the correct (i.e., parent) zone. This is done by ensuring that the SOA bit is not set in the Type Bit Maps field of this NSEC4 RR.

Note that the presence of an NS bit implies the absence of a DNAME bit, so there is no need to check for the DNAME bit in the Type Bit Maps field of the NSEC4 RR.

If there is no NSEC4 RR present that matches the delegation name, then the validator MUST verify that there is a closest provable encloser for the delegation name. In addition, the validator MUST verify that there is an NSEC4 RR that covers the next closer name and has the Opt-Out bit set.

9. Resolver Considerations

9.1. NSEC4 Resource Record Caching

The same considerations as described in Section 9.1 of [RFC5155] for NSEC3 apply to NSEC4.

9.2. Use of the AD Bit

The same considerations as described in Section 9.2 of [RFC5155] for NSEC3 apply to NSEC4.

10. Special Considerations

10.1. Domain Name Length Restrictions

The same considerations as described in Section 10.1 of [RFC5155] apply.

10.2. DNAME at the Zone Apex

The DNAME specification in Section 3 of [RFC6672] has a 'no-descendants' limitation. If a DNAME RR is present at node N, there MUST be no data at any descendant of N.

[RFC5155] updates the DNAME specification to allow NSEC3 and RRSIG types at descendants of the apex regardless of the existence of DNAME at the apex.

This document updates the DNAME specification to also allow NSEC4 types at descendants of the apex regardless of the existence of DNAME at the apex.

10.3. Iterations value

Like Section 10.3 in [RFC5155], but we recommend to read [Schaeffer10] as it shows that a lower iterations value is also acceptable. The research shows that that the half performance count for validators is roughly 150 to 600, depending on the key size. For authoritative servers the half performance count is around 100 iterations.

10.4. More Special Considerations

Appendix C of [RFC5155] clarifies specific behavior and explains more special considerations for implementations, regarding salting and hash collisions. These considerations for NSEC3 also apply to NSEC4.

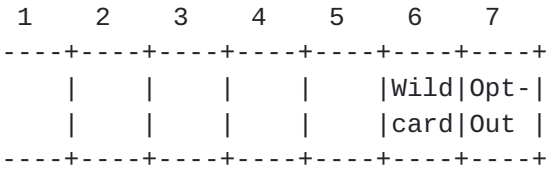
11. IANA Considerations

Although the NSEC4 and NSEC4PARAM RR formats include a hash algorithm parameter, this document does not define a particular mechanism for safely transitioning from one NSEC4 hash algorithm to another. When specifying a new hash algorithm for use with NSEC4, a transition mechanism MUST also be defined.

This document updates the IANA registry "DOMAIN NAME SYSTEM PARAMETERS" (http://www.iana.org/assignments/dns-parameters) in sub-registry "TYPES", by defining two new types. Section 3 defines the NSEC4 RR type [TBD]. Section 4 defines the NSEC4PARAM RR type [TBD].

This document possibly updates the IANA registry "DNS SECURITY ALGORITHM NUMBERS -- per [RFC4035]" (http://www.iana.org/assignments/dns-sec-alg-numbers).

This document creates a new IANA registry for NSEC4 flags. This registry is named "DNSSEC NSEC4 Flags". The initial contents of this registry are:



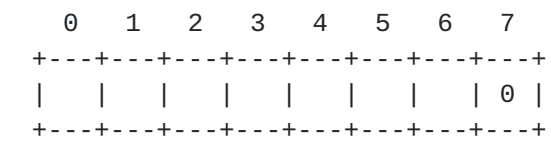
bit 6 is the Wildcard flag.

bit 7 is the Opt-Out flag.

bits 0 - 5 are available for assignment.

Assignment of additional NSEC4 Flags in this registry requires IETF Standards Action [RFC5226].

This document creates a new IANA registry for NSEC4PARAM flags. This registry is named "DNSSEC NSEC4PARAM Flags". The initial contents of this registry are:



bit 7 is reserved and must be 0.

bits 0 - 6 are available for assignment.

Assignment of additional NSEC4PARAM Flags in this registry requires IETF Standards Action [RFC5226].

Finally, this document creates a new IANA registry for NSEC4 hash algorithms. This registry is named "DNSSEC NSEC4 Hash Algorithms". The initial contents of this registry are:

0 is the Identity function.

1 is SHA-1.

2-255 Available for assignment.

Assignment of additional NSEC4 hash algorithms in this registry requires IETF Standards Action [RFC5226].

12. Security Considerations

This document does not introduce any new security issues beyond those already discussed in [RFC4033], [RFC4034], [RFC4035] and [RFC5155].

13. Acknowledgements

This document would not be possible without the help of Ed Lewis, Roy Arends, Wouter Wijngaards, Karst Koymans, Mohan Parthasarathy, Marco Davids, Esther Makaay and Antoin Verschuren.

This document was produced using the xml2rfc tool ([RFC2629]) and Pandoc2rfc ([Gieben11]).

14. Changelog

14.1. 01

- Clarification throughout the text (Mohan Parthasarathy);
- Add section about empty non-terminals in NSEC, NSEC3 and NSEC4;

- Rename Zero hashing to Identity function.
- No need for different ordering mechanisms: canonical ordering only.
- Remove section on validator algorithm (already explained in RFC4035).

14.2. 00

- Initial document.

15. References

15.1. Informative References

[Gieben11] Gieben, R., "Pandoc2RFC", September 2011, <https://github.com/miekg/pandoc2rfc/>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, July 2006.

[RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, June 2012.

15.2. Normative References

[GiebenMekking11] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", January 2012, <http://www.sidn.nl/fileadmin/docs/PDF-files_UK/wp-2011-0x01-v2.pdf>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS

UPDATE)", RFC 2136, April 1997.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, September 2003.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4955] Blacka, D., "DNS Security (DNSSEC) Experiments", RFC 4955, July 2007.

[RFC4956] Arends, R., Kosters, M., and D. Blacka, "DNS Security (DNSSEC) Opt-In", RFC 4956, July 2007.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[Schaeffer10] Schaeffer, Y., "NSEC3 Hash Performance", March 2010, <http://www.nlnetlabs.nl/downloads/publications/nsec3_hash_performance.pdf>.

Appendix A. List of Hashed Owner Names

The following owner names are used in this document. The hashed names are hashed with SHA1 using an empty salt and zero iterations.

Original Name	Hashed Name
example.	3msev9usmd4br9s97v51r2tdvmr9iqo1
a.example.	6cd522290vma0nr8lqu1ivtcofj94rga
ns1.example.	m1o89lfdo9rrf2f8r8ss42d81d09v48m
sd.example.	831naajdsm14h0md3kip92563ud3saav
ns1.sd.example.	qrsbil3cs97oa4p5fql8dedp6jo0b9a6
ud.example.	ub8e42kj4s2jdfve6aloo98jdoa425a9
ns1.ud.example.	7cuee8ri909f5r365jqr0k6j75thndpi
who.example.	g4s20q3kptookhpt9mgr93k8bfhjs3fd
*.who.example.	ht6ocje68mtm96jpes8olrlbf67jjvdu
b.who.example.	rmv5tauk8nss83vo1st0tp1ps927j71e

Appendix B. Example Zones

B.1. Hashed Denial of Existence

This is the unsigned zone we are using for the NSEC4 examples. The overall TTL and class are left out for clarity.

```
$ORIGIN example.
@               SOA     ns1.example. bugs.example. 1 2 3 4 5
                NS      ns1.example.
ns1             A       192.0.2.10
;; secure delegation
sd              NS      ns1.sd.example.
                DS      33694 253 2 ...
ns1.sd          A       192.0.2.10
;; unsecure delegation
ud              NS      ns1.ud.example.
ns1.ud          A       192.0.2.10
*.who           TXT     "Wildcard"
```

B.2. Identity Function

This is the same zone shown with the Identity function. The RRSIG Signature field, the DNSKEY Public Key field and the DS Digest field are omitted. The RRSIG expiration and inception times are set to ".".

```
$ORIGIN example.
@               SOA     ns1.example. bugs.example. 1 2 3 4 5
                RRSIG   SOA 253 1 300 . . 39824 example. ...
                RRSIG   NS 253 1 300 . . 39824 example. ...
                RRSIG   DNSKEY 253 1 300 . . 39824 example. ...
                RRSIG   NSEC4PARAM 253 1 3600 . . 39824 example. ...
                RRSIG   NSEC4 253 1 5 . . 39824 example. ...
                NS      ns1.example.
                DNSKEY  256 3 253 ...
                NSEC4PARAM 0 0 0 -
                NSEC4   0 1 0 - (
                ns1.example. NS SOA RRSIG DNSKEY NSEC4 NSEC4PARAM )
ns1             A       192.0.2.10
                RRSIG   A 253 2 300 . . 39824 example. ...
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
                NSEC4   0 1 0 - sd.example. A RRSIG NSEC4
sd              NS      ns1.sd.example.
                DS      33694 253 2 ...
                RRSIG   DS 253 2 300 . . 39824 example. ...
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
                NSEC4   0 1 0 - who.example. NS DS RRSIG NSEC4
ns1.sd          A       192.0.2.10
ud              NS      ns1.ud.example.
ns1.ud          A       192.0.2.10
who             NSEC4   0 3 0 - *.who.example.
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
*.who           TXT     "Wildcard"
                RRSIG   TXT 253 2 300 . . 39824 example. ...
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
                NSEC4   0 1 0 - example. TXT RRSIG NSEC4
```

B.3. SHA1 Hashing

This is the same zone shown with SHA1 hashing.

```
$ORIGIN example.
@               SOA     ns1.example. bugs.example. 1 2 3 4 5
                RRSIG   SOA 253 1 300 . . 39824 example. ...
                RRSIG   NS 253 1 300 . . 39824 example. ...
                RRSIG   DNSKEY 253 1 300 . . 39824 example. ...
                RRSIG   NSEC4PARAM 253 1 3600 . . 39824 example. ...
                NS      ns1.example.
                DNSKEY  256 3 253 ...
                NSEC4PARAM 1 0 0 -
3msev9usmd4br9s97v51r2tdvmr9iqo1 NSEC4 1 1 0 - (
                831naajdsm14h0md3kip92563ud3saav.example.
                NS SOA RRSIG DNSKEY NSEC4PARAM )
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
831naajdsm14h0md3kip92563ud3saav NSEC4 1 1 0 - (
                g4s20q3kptookhpt9mgr93k8bfhjs3fd.example.
                NS DS RRSIG )
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
g4s20q3kptookhpt9mgr93k8bfhjs3fd NSEC4 1 3 0 - (
                ht6ocje68mtm96jpes8olrlbf67jjvdu.example. )
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
ht6ocje68mtm96jpes8olrlbf67jjvdu NSEC4 1 1 0 - (
                m1o89lfdo9rrf2f8r8ss42d81d09v48m.example.
                TXT RRSIG )
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
m1o89lfdo9rrf2f8r8ss42d81d09v48m NSEC4 1 1 0 - (
                3msev9usmd4br9s97v51r2tdvmr9iqo1.example.
                A RRSIG )
                RRSIG   NSEC4 253 2 5 . . 39824 example. ...
ns1             A       192.0.2.10
                RRSIG   A 253 2 300 . . 39824 example. ...
sd              NS      ns1.sd.example.
                DS      33694 253 2 ...
                RRSIG   DS 253 2 300 . . 39824 example. ...
ns1.sd          A       192.0.2.10

ud              NS      ns1.ud.example.
ns1.ud          A       192.0.2.10
*.who           TXT     "Wildcard"
                RRSIG   TXT 253 2 300 . . 39824 example. ...
```

Appendix C. Example Responses

The examples in this section show response messages using the signed zone example in Appendix B.3.

C.1. Name Error

An authoritative name error. The NSEC4 RRs prove that the name does not exist and that there is no wildcard RR that should have been expanded.

```
;; Header: QR AA RD RCODE=NXDOMAIN
;;
;; Question
a.example.      IN A

;; Answer
;; (empty)

;; Authority
;; NSEC4 RR that matches the closest encloser (example)
;; This NSEC4 also covers the next closer name (a.example)
;; H(a.example) = 6cd522290vma0nr8lqu1ivtcofj94rga
3msev9usmd4br9s97v51r2tdvmr9iqo1.example. NSEC4 1 1 0 - (
                831naajdsm14h0md3kip92563ud3saav.example.
                NS SOA RRSIG DNSKEY NSEC4PARAM )
3msev9usmd4br9s97v51r2tdvmr9iqo1.example. RRSIG NSEC4 253 2 5 (
                . . 39824 example. ... )
example.        SOA     ns1.example. bugs.example. 1 2 3 4 5
example.        RRSIG   SOA 253 1 300 . . 39824 example. ...
```

The query returns one NSEC4 RR that proves that the requested data does not exist and that no wildcard expansion applies. The negative response is authenticated by verifying the NSEC4 RR. The corresponding RRSIGs indicate that the NSEC4 RRs are signed by an "example" DNSKEY of algorithm 253 and with key tag 39824. The resolver needs the corresponding DNSKEY RR in order to authenticate this answer.

In the above example, the name "example" hashes to "3msev9usmd4br9s97v51r2tdvmr9iqo1". This indicates that this might be the closest encloser.

To prove that "a.example" does not exist, the name is hashed to "6cd522290vma0nr8lqu1ivtcofj94rga". The NSEC4 RR also proves that next closer name does not exist.

To prove that the source of synthesis "*.example" does not exist, the Wildcard bit at the NSEC4 RR matching the closest encloser is inspected. The bit is clear and this shows that the source of synthesis does indeed not exist.

C.2. No Data Error

A No Data Response. The NSEC4 RR proves that the name exists and that the requested RR type does not.

```
;; Header: QR AA RD RCODE=NOERROR
;;
;; Question
ns1.example.    IN MX

;; Answer
;; (empty)

;; Authority
example.        SOA     ns1.example. bugs.example. 1 2 3 4 5
example.        RRSIG   SOA 253 1 300 . . 39824 example. ...
;; H(ns1.example) = m1o89lfdo9rrf2f8r8ss42d81d09v48m
m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. NSEC4 1 1 0 - (
                3msev9usmd4br9s97v51r2tdvmr9iqo1.example.
                A RRSIG )
m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. RRSIG NSEC4 253 2 5 (
                . . 39824 example. ... )
```

The query returned an NSEC4 RR that proves that the requested name exists ("ns1.example" hashes to "m1o89lfdo9rrf2f8r8ss42d81d09v48m"), but the requested RR type does not exist (type MX is absent in the type code list of the NSEC4 RR), and was not a CNAME (type CNAME is also absent in the type code list of the NSEC4 RR).

C.3. Referral to an Opt-Out Unsigned Zone

The NSEC4 RRs prove that nothing for this delegation was signed. There is no proof that the unsigned delegation exists.

```
;; Header: QR RD RCODE=NOERROR
;;
;; Question
a.ud.example.   IN MX

;; Answer
;; (empty)

;; Authority
ud.example.     NS      ns1.ud.example.

;; NSEC4 RR that matches the closest provable encloser (example)
;; H(example) = 3msev9usmd4br9s97v51r2tdvmr9iqo1
3msev9usmd4br9s97v51r2tdvmr9iqo1.example. NSEC4 1 1 0 - (
                831naajdsm14h0md3kip92563ud3saav.example.
                NS SOA RRSIG DNSKEY NSEC4PARAM )
3msev9usmd4br9s97v51r2tdvmr9iqo1.example. RRSIG NSEC4 253 2 5 (
                . . 39824 example. ... )

;; NSEC4 RR that covers the next closer name (ud.example)
;; H(ud.example) = ub8e42kj4s2jdfve6aloo98jdoa425a9
m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. NSEC4 1 1 0 - (
                3msev9usmd4br9s97v51r2tdvmr9iqo1.example.
                A RRSIG )
m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. RRSIG NSEC4 253 2 5 (
                . . 39824 example. ... )

;; Additional
ns1.ud.example. A       192.0.2.10
```

The query returned a referral to the unsigned "ud.example." zone. The response contains the closest provable encloser of "ud.example" to be "example", since the hash of "ud.example" ("ub8e42kj4s2jdfve6aloo98jdoa425a9") is covered by the first NSEC4 RR and its Opt-Out bit is set.

C.4. Wildcard Expansion

A query that was answered with a response containing a wildcard expansion. The label count in the RRSIG RRSet in the answer section indicates that a wildcard RRSet was expanded to produce this response, and the NSEC4 RR proves that no next closer name exists in the zone.

```
;; Header: QR AA RD RCODE=NOERROR
;;
;; Question
a.b.who.example.        IN TXT

;; Answer
a.b.who.example.        TXT     "Wildcard"
a.b.who.example. RRSIG TXT 253 2 300 (
                . . 39824 example. ... )

;; Authority
;; NSEC4 RR that covers the next closer name (b.who.example)
;; H(b.who.example) = rmv5tauk8nss83vo1st0tp1ps927j71e
m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. NSEC4 1 1 0 - (
                3msev9usmd4br9s97v51r2tdvmr9iqo1.example.
                A RRSIG )
m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. RRSIG NSEC4 253 2 5 (
                . . 39824 example. ... )
example.        NS      ns1.example.
example.        RRSIG   NS 253 1 300 . . 39824 example. ...

;; Additional
ns1.example.    A       192.0.2.10
ns1.example.    RRSIG   A 253 2 300 . . 39824 example. ...
```

The query returned an answer that was produced as a result of a wildcard expansion. The answer section contains a wildcard RRSet expanded as it would be in a traditional DNS response. The RRSIG Labels field value of 2 indicates that the answer is the result of a wildcard expansion, as the "a.b.who.example" name contains 4 labels. This also shows that "who.example" exists, so there is no need for an NSEC4 RR that matches the closest encloser.

The NSEC4 RR proves that no closer match could have been used to answer this query.

C.5. Wildcard No Data Error

A No Data Response for a name covered by a wildcard. The NSEC4 RRs prove that the matching wildcard name does not have any RRs of the requested type and that no closer match exists in the zone.

```
   ;; Header: QR AA RD RCODE=NOERROR
   ;;
   ;; Question
   a.b.who.example.        IN AAAA

   ;; Answer
   ;; (empty)

   ;; Authority
   ;; NSEC4 RR that covers the next closer name (b.who.example)
   ;; H(b.who.example) = rmv5tauk8nss83vo1st0tp1ps927j71e
   m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. NSEC4 1 1 0 - (
                  3msev9usmd4br9s97v51r2tdvmr9iqo1.example.
                  A RRSIG )
   m1o89lfdo9rrf2f8r8ss42d81d09v48m.example. RRSIG NSEC4 253 2 5 (
                  . . 39824 example. ... )
   example.        SOA     ns1.example. bugs.example. 1 2 3 4 5
   example.        RRSIG   SOA 253 1 300 . . 39824 example. ...
   ;; NSEC4 RR that matches the wildcard at closest encloser
   ;; H(*.who.example) = ht6ocje68mtm96jpes8olrlbf67jjvdu
   ht6ocje68mtm96jpes8olrlbf67jjvdu.example. NSEC4 1 1 0 - (
                  m1o89lfdo9rrf2f8r8ss42d81d09v48m.example.
                  TXT RRSIG )
   ht6ocje68mtm96jpes8olrlbf67jjvdu.example. RRSIG NSEC4 253 2 5 (
                  . . 39824 example. ... )
```

The query returned the NSEC4 RRs that prove that the requested data does not exist and shows the types that do exist at the wildcard.

Authors' Addresses

R. (Miek) Gieben
SIDN Labs
Meander 501
Arnhem, 6825 MD
NL

Phone:
EMail: miek.gieben@sidn.nl
URI: https://sidn.nl/

W. (Matthijs) Mekking
NLnet Labs
Science Park 400
Amsterdam, 1098 XH
NL

Phone:
EMail: matthijs@nlnetlabs.nl
URI: http://www.nlnetlabs.nl/