Network Working Group
Internet-Draft
Expires: August 5, 2006

B. Laurie
G. Sisson
R. Arends
Nominet
February 2006

DNSSEC Hash Authenticated Denial of Existence
draft-ietf-dnsext-nsec3-04

Status of this Memo

Copyright Notice

Abstract

The DNS Security Extensions introduces the NSEC resource record for authenticated denial of existence. This document introduces a new resource record as an alternative to NSEC that provides measures against zone enumeration and allows for gradual expansion of delegation-centric zones.

Table of Contents

1. Introduction

1.1. Rationale

The DNS Security Extensions included the NSEC RR to provide authenticated denial of existence. Though the NSEC RR meets the requirements for authenticated denial of existence, it introduced a side-effect in that the contents of a zone can be enumerated. This property introduces undesired policy issues.

An enumerated zone can be used either directly as a source of probable e-mail addresses for spam, or indirectly as a key for multiple WHOIS queries to reveal registrant data which many registries may be under strict legal obligations to protect. Many registries therefore prohibit copying of their zone file; however the use of NSEC RRs renders these policies unenforceable.

A second problem was the requirement that the existence of all record types in a zone - including unsigned delegation points - must be accounted for, despite the fact that unsigned delegation point records are not signed. This requirement has a side-effect that the overhead of signed zones is not related to the increase in security of subzones. This requirement does not allow the zones' size to grow in relation to the growth of signed subzones.

In the past, solutions (draft-ietf-dnsext-dnssec-opt-in) have been proposed as a measure against these side effects but at the time were regarded as secondary over the need to have a stable DNSSEC specification. With (draft-vixie-dnssec-ter) [14] a graceful

transition path to future enhancements is introduced, while current DNSSEC deployment can continue. This document presents the NSEC3 Resource Record which mitigates these issues with the NSEC RR.

The reader is assumed to be familiar with the basic DNS and DNSSEC concepts described in RFC 1034 [1], RFC 1035 [2], RFC 4033 [3], RFC 4034 [4], RFC 4035 [5] and subsequent RFCs that update them: RFC 2136 [6], RFC2181 [7] and RFC2308 [8].

1.2. Reserved Words

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 RFC 2119 [9].

1.3. Terminology

The practice of discovering the contents of a zone, i.e. enumerating the domains within a zone, is known as "zone enumeration". Zone

enumeration was not practical prior to the introduction of DNSSEC.

In this document the term "original ownername" refers to a standard ownername. Because this proposal uses the result of a hash function over the original (unmodified) ownername, this result is referred to as "hashed ownername".

"Hash order" means the order in which hashed ownernames are arranged according to their numerical value, treating the leftmost (lowest numbered) octet as the most significant octet. Note that this is the same as the canonical ordering specified in RFC 4034 [4].

An "empty non-terminal" is a domain name that owns no resource records but has subdomains that do.

The "closest encloser" of a (nonexistent) domain name is the longest domain name, including empty non-terminals, that matches the rightmost part of the nonexistent domain name.

"Base32 encoding" is "Base 32 Encoding with Extended Hex Alphabet" as specified in RFC 3548bis [15].

2. NSEC versus NSEC3

This document does NOT obsolete the NSEC record, but gives an alternative for authenticated denial of existence. NSEC and NSEC3 RRs can not co-exist in a zone. See draft-vixie-dnssec-ter [14] for a signaling mechanism to allow for graceful transition towards NSEC3.

3. The NSEC3 Resource Record

The NSEC3 RR provides Authenticated Denial of Existence for DNS Resource Record Sets.

The NSEC3 Resource Record (RR) lists RR types present at the NSEC3 RR's original ownername. It includes the next hashed ownername in the hash order of the zone. The complete set of NSEC3 RRs in a zone indicates which RRsets exist for the original ownername of the RRset and form a chain of hashed ownernames in the zone. This information is used to provide authenticated denial of existence for DNS data, as described in RFC 4035 [5]. To provide protection against zone enumeration, the ownernames used in the NSEC3 RR are cryptographic hashes of the original ownername prepended to the name of the zone. The NSEC3 RR indicates which hash function is used to construct the hash, which salt is used, and how many iterations of the hash function are performed over the original ownername. The hashing

technique is described fully in Section 5.

Hashed ownernames of unsigned delegations may be excluded from the chain. An NSEC3 record which span covers the hash of an unsigned delegation's ownername is referred to as an Opt-In NSEC3 record and is indicated by the presence of a flag.

The ownername for the NSEC3 RR is the base32 encoding of the hashed ownername prepended to the name of the zone..

The type value for the NSEC3 RR is XX.

The NSEC3 RR RDATA format is class independent and is described below.

The class MUST be the same as the original ownername's class.

The NSEC3 RR SHOULD have the same TTL value as the SOA minimum TTL field. This is in the spirit of negative caching [8].

3.1. NSEC3 RDATA Wire Format

The RDATA of the NSEC3 RR is as shown below:

"O" is the Opt-In Flag field.

3.1.1. The Hash Function Field

The Hash Function field identifies the cryptographic hash function used to construct the hash-value.

The values are as defined for the DS record (see RFC 3658 [10]).

On reception, a resolver MUST ignore an NSEC3 RR with an unknown hash function value.

3.1.2. The Opt-In Flag Field

The Opt-In Flag field indicates whether this NSEC3 RR covers unsigned delegations.

In DNSSEC, NS RRsets at delegation points are not signed, and may be accompanied by a DS record. The security status of the subzone is determined by the presence or absence of the DS RRset,

cryptographically proven by the NSEC record or the signed DS RRset. The presence of the Opt-In flag expands this definition by allowing insecure delegations to exist within an otherwise signed zone without the corresponding NSEC3 record at the delegation's (hashed) owner name. These delegations are proven insecure by using a covering NSEC3 record.

Resolvers must be able to distinguish between NSEC3 records and Opt-In NSEC3 records. This is accomplished by setting the Opt-In flag of the NSEC3 records that cover (or potentially cover) insecure delegation nodes.

An Opt-In NSEC3 record does not assert the existence or non-existence of the insecure delegations that it covers. This allows for the addition or removal of these delegations without recalculating or resigning records in the NSEC3 chain. However, Opt-In NSEC3 records do assert the (non)existence of other, authoritative RRsets.

An Opt-In NSEC3 record MAY have the same original owner name as an insecure delegation. In this case, the delegation is proven insecure by the lack of a DS bit in type map and the signed NSEC3 record does assert the existence of the delegation.

Zones using Opt-In MAY contain a mixture of Opt-In NSEC3 records and non-Opt-In NSEC3 records. If an NSEC3 record is not Opt-In, there MUST NOT be any hashed ownernames of insecure delegations (nor any other records) between it and the RRsets indicated by the 'Next Hashed Ownername' in the NSEC3 RDATA. If it is Opt-In, there MUST only be hashed ownernames of insecure delegations between it and the next node indicated by the 'Next Hashed Ownername' in the NSEC3 RDATA.

In summary,

- An Opt-In NSEC3 type is identified by an Opt-In Flag field value of 1.
- A non Opt-In NSEC3 type is identified by an Opt-In Flag field value of 0.

and,

- An Opt-In NSEC3 record does not assert the non-existence of a hash

ownername between its ownername and next hashed ownername, although it does assert that any hashed name in this span MUST be of an insecure delegation.

- An Opt-In NSEC3 record does assert the (non)existence of RRsets with the same hashed owner name.

3.1.3. The Iterations Field

The Iterations field defines the number of times the hash has been iterated. More iterations results in greater resiliency of the hash value against dictionary attacks, but at a higher cost for both the server and resolver. See Section 5 for details of this field's use.

Iterations make an attack more costly by making the hash computation more computationally intensive, e.g. by iterating the hash function a number of times.

When generating a few hashes this performance loss will not be a problem, as a validator can handle a delay of a few milliseconds. But when doing a dictionary attack it will also multiply the attack workload by a factor, which is a problem for the attacker.

3.1.4. The Salt Length Field

The salt length field defines the length of the salt in octets.

3.1.5. The Salt Field

The Salt field is not present when the Salt Length Field has a value of 0.

The Salt field is appended to the original ownername before hashing in order to defend against precalculated dictionary attacks. See Section 5 for details on how the salt is used.

Salt is used to make dictionary attacks using precomputation more costly. A dictionary can only be computed after the attacker has the salt, hence a new salt means that the dictionary has to be regenerated with the new salt.

There MUST be a complete set of NSEC3 records covering the entire zone that use the same salt value. The requirement exists so that, given any qname within a zone, at least one covering NSEC3 RRset may be found. While it may be theoretically possible to produce a set of NSEC3s that use different salts that cover the entire zone, it is computationally infeasible to generate such a set. See Section 8.2 for further discussion.

The salt value SHOULD be changed from time to time - this is to prevent the use of a precomputed dictionary to reduce the cost of enumeration.

3.1.6. The Next Hashed Ownername Field

The Next Hashed Ownername field contains the next hashed ownername in hash order. That is, given the set of all hashed owernames, the Next Hashed Ownername contains the hash value that immediately follows the owner hash value for the given NSEC3 record. The value of the Next Hashed Ownername Field in the last NSEC3 record in the zone is the same as the ownername of the first NSEC3 RR in the zone in hash order.

Hashed ownernames of glue RRsets MUST NOT be listed in the Next Hashed Ownername unless at least one authoritative RRset exists at the same ownername. Hashed ownernames of delegation NS RRsets MUST be listed if the Opt-In bit is clear.

Note that the Next Hashed Ownername field is not encoded, unlike the NSEC3 RR's ownername. It is the unmodified binary hash value. It does not include the name of the containing zone.

The length of this field is the length of the hash value produced by the hash function selected by the Hash Function field.

3.1.7. The Type Bit Maps Field

The Type Bit Maps field identifies the RRset types which exist at the NSEC3 RR's original ownername.

The Type bits for the NSEC3 RR and RRSIG RR MUST be set during generation, and MUST be ignored during processing.

The RR type space is split into 256 window blocks, each representing the low-order 8 bits of the 16-bit RR type space. Each block that has at least one active RR type is encoded using a single octet window number (from 0 to 255), a single octet bitmap length (from 1 to 32) indicating the number of octets used for the window block's bitmap, and up to 32 octets (256 bits) of bitmap.

Blocks are present in the NSEC3 RR RDATA in increasing numerical order.

"|" denotes concatenation

```
   Type Bit Map(s) Field = ( Window Block # | Bitmap Length | Bitmap ) +
```


Each bitmap encodes the low-order 8 bits of RR types within the window block, in network bit order. The first bit is bit 0. For window block 0, bit 1 corresponds to RR type 1 (A), bit 2 corresponds to RR type 2 (NS), and so forth. For window block 1, bit 1 corresponds to RR type 257, bit 2 to RR type 258. If a bit is set to 1, it indicates that an RRset of that type is present for the NSEC3 RR's ownername. If a bit is set to 0, it indicates that no RRset of that type is present for the NSEC3 RR's ownername.

Since bit 0 in window block 0 refers to the non-existing RR type 0, it MUST be set to 0. After verification, the validator MUST ignore the value of bit 0 in window block 0.

Bits representing Meta-TYPEs or QTYPEs as specified in RFC 2929 [11] (section 3.1) or within the range reserved for assignment only to QTYPEs and Meta-TYPEs MUST be set to 0, since they do not appear in zone data. If encountered, they must be ignored upon reading.

Blocks with no types present MUST NOT be included. Trailing zero octets in the bitmap MUST be omitted. The length of each block's bitmap is determined by the type code with the largest numerical value, within that block, among the set of RR types present at the NSEC3 RR's actual ownername. Trailing zero octets not specified MUST be interpreted as zero octets.

3.2. The NSEC3 RR Presentation Format

The presentation format of the RDATA portion is as follows:

The Opt-In Flag Field is represented as an unsigned decimal integer. The value is either 0 or 1.

The Hash field is presented as a mnemonic of the hash or as an unsigned decimal integer. The value has a maximum of 127.

The Iterations field is presented as an unsigned decimal integer.

The Salt Length field is not presented.

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 value 0.

The Next Hashed Ownername field is represented as a sequence of case-insensitive base32 digits, without whitespace.

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 RFC 3597 [12] (section 5) MUST be used.

4. Creating Additional NSEC3 RRs for Empty Non-Terminals

In order to prove the non-existence of a record that might be covered by a wildcard, it is necessary to prove the existence of its closest encloser. A closest encloser might be an empty non-terminal.

Additional NSEC3 RRs are generated for empty non-terminals. These additional NSEC3 RRs are identical in format to NSEC3 RRs that cover existing RRs in the zone except that their type-maps only indicated the existence of an NSEC3 RRset and an RRSIG RRset.

This relaxes the requirement in Section 2.3 of RFC4035 that NSEC RRs not appear at names that did not exist before the zone was signed. [Comment.1]

5. Calculation of the Hash

Define H(x) to be the hash of x using the hash function selected by the NSEC3 record and || to indicate concatenation. Then define:

IH(salt,x,0)=H(x || salt)

IH(salt,x,k)=H(IH(salt,x,k-1) || salt) if k > 0

Then the calculated hash of an ownername is IH(salt,ownername,iterations-1), where the ownername is the canonical form.

The canonical form of the ownername is the wire format of the ownername where:

1. The ownername is fully expanded (no DNS name compression) and fully qualified;
2. All uppercase US-ASCII letters are replaced by the corresponding lowercase US-ASCII letters;
3. If the ownername is a wildcard name, the ownername is in its original unexpanded form, including the "*" label (no wildcard substitution);

This form is as defined in section 6.2 of RFC 4034 ([4]).

6. Including NSEC3 RRs in a Zone

Each ownername within the zone that owns authoritative RRsets MUST

have a corresponding NSEC3 RR. Ownernames that correspond to unsigned delegations MAY have a corresponding NSEC3 RR, however, if there is not, there MUST be a covering NSEC3 RR with the Opt-In flag set to 1. Other non-authoritative RRs are not included in the set of NSEC3 RRs.

Each empty non-terminal MUST have an NSEC3 record.

The TTL value for any NSEC3 RR SHOULD be the same as the minimum TTL value field in the zone SOA RR.

The type bitmap of every NSEC3 resource record in a signed zone MUST indicate the presence of both the NSEC3 RR type itself and its corresponding RRSIG RR type.

The following steps describe the proper construction of NSEC3 records. [Comment.2]

1. For each unique original ownername in the zone, add an NSEC3 RRset. If Opt-In is being used, ownernames of unsigned delegations may be excluded, but must be considered for empty-non-terminals. The ownername of the NSEC3 RR is the hashed equivalent of the original owner name, prepended to the zone name. The Next Hashed Ownername field is left blank for the moment. If Opt-In is being used, set the Opt-In bit to one.
2. For each RRset at the original owner name, set the corresponding

bit in the type bit map.

3. If the difference in number of labels between the apex and the original ownername is greater then 1, additional NSEC3s need to be added for every empty non-terminal between the apex and the original ownername. This process may generate NSEC3 RRs with duplicate hashed ownernames.
4. Sort the set of NSEC3 RRs into hash order. Hash order is the ascending numerical order of the non-encoded hash values.
5. Combine NSEC3 RRs with identical hashed ownernames by replacing with a single NSEC3 RR with the type map consisting of the union of the types represented by the set of NSEC3 RRs.
6. In each NSEC3 RR, insert the Next Hashed Ownername by using the value of the next NSEC3 RR in hash order. The Next Hashed Ownername of the last NSEC3 in the zone contains the value of the hashed ownername of the first NSEC3 in the hash order.

7. Responding to NSEC3 Queries

Since NSEC3 ownernames are not represented in the NSEC3 chain like other zone ownernames, direct queries for NSEC3 ownernames present a special case.

The special case arises when the following are all true:

- The QNAME equals an existing NSEC3 ownername, and
- There are no other record types that exist at QNAME, and
- The QTYPE does not equal NSEC3.

These conditions describe a particular case: the answer should be a NOERROR/NODATA response, but there is no NSEC3 RRset for H(QNAME) to include in the authority section.

However, the NSEC3 RRset with ownername equal to QNAME is able to prove its own existence. Thus, when answering this query, the authoritative server MUST include the NSEC3 RRset whose ownername equals QNAME. This RRset proves that QNAME is an existing name with types NSEC3 and RRSIG. The authoritative server MUST also include the NSEC3 RRset that covers the hash of QNAME. This RRset proves that no other types exist.

When validating a NOERROR/NODATA response, validators MUST check for a NSEC3 RRset with ownername equals to QNAME, and MUST accept that

(validated) NSEC3 RRset as proof that QNAME exists. The validator MUST also check for an NSEC3 RRset that covers the hash of QNAME as proof that QTYPE doesn't exist.

Other cases where the QNAME equals an existing NSEC3 ownername may be answered normally.

8. Special Considerations

The following paragraphs clarify specific behaviour explain special considerations for implementations.

8.1. Proving Nonexistence

If a wildcard resource record appears in a zone, its asterisk label is treated as a literal symbol and is treated in the same way as any other ownername for purposes of generating NSEC3 RRs. RFC 4035 [5] describes the impact of wildcards on authenticated denial of existence.

In order to prove there exist no RRs for a domain, as well as no source of synthesis, an RR must be shown for the closest encloser, and non-existence must be shown for all closer labels and for the wildcard at the closest encloser.

This can be done as follows. If the QNAME in the query is omega.alfa.beta.example, and the closest encloser is beta.example (the nearest ancestor to omega.alfa.beta.example), then the server should return an NSEC3 that demonstrates the nonexistence of

alfa.beta.example, an NSEC3 that demonstrates the nonexistence of *.beta.example, and an NSEC3 that demonstrates the existence of beta.example. This takes between one and three NSEC3 records, since a single record can, by chance, prove more than one of these facts.

When a verifier checks this response, then the existence of beta.example together with the non-existence of alfa.beta.example proves that the closest encloser is indeed beta.example. The non-existence of *.beta.example shows that there is no wildcard at the closest encloser, and so no source of synthesis for omega.alfa.beta.example. These two facts are sufficient to satisfy

the resolver that the QNAME cannot be resolved.

In practice, since the NSEC3 owner and next names are hashed, if the server responds with an NSEC3 for beta.example, the resolver will have to try successively longer names, starting with example, moving to beta.example, alfa.beta.example, and so on, until one of them hashes to a value that matches the interval (but not the ownername nor next owner name) of one of the returned NSEC3s (this name will be alfa.beta.example). Once it has done this, it knows the closest encloser (i.e. beta.example), and can then easily check the other two required proofs.

Note that it is not possible for one of the shorter names tried by the resolver to be denied by one of the returned NSEC3s, since, by definition, all these names exist and so cannot appear within the range covered by an NSEC3. Note, however, that the first name that the resolver tries MUST be the apex of the zone, since names above the apex could be denied by one of the returned NSEC3s.

8.2. Salting

Augmenting original ownernames with salt before hashing increases the cost of a dictionary of pre-generated hash-values. For every bit of salt, the cost of a precomputed dictionary doubles (because there must be an entry for each word combined with each possible salt value). The NSEC3 RR can use a maximum of 2040 bits (255 octets) of salt, multiplying the cost by 2^2040. This means that an attacker must, in practice, recompute the dictionary each time the salt is changed.

There MUST be at least one complete set of NSEC3s for the zone using the same salt value.

The salt SHOULD be changed periodically to prevent precomputation using a single salt. It is RECOMMENDED that the salt be changed for every resigning.

Note that this could cause a resolver to see records with different salt values for the same zone. This is harmless, since each record stands alone (that is, it denies the set of ownernames whose hashes, using the salt in the NSEC3 record, fall between the two hashes in

the NSEC3 record) - it is only the server that needs a complete set of NSEC3 records with the same salt in order to be able to answer every possible query.

There is no prohibition with having NSEC3 with different salts within the same zone. However, in order for authoritative servers to be able to consistently find covering NSEC3 RRs, the authoritative server MUST choose a single set of parameters (algorithm, salt, and iterations) to use when selecting NSEC3s. In the absence of any other metadata, the server does this by using the parameters from the zone apex NSEC3, recognizable by the presence of the SOA bit in the type map. If there is more than one NSEC3 record that meets this description, then the server may arbitrarily choose one. Because of this, if there is a zone apex NSEC3 RR within a zone, it MUST be part of a complete NSEC3 set. Conversely, if there exists an incomplete set of NSEC3 RRs using the same parameters within a zone, there MUST NOT be an NSEC3 RR using those parameters with the SOA bit set.

8.3. Iterations

Setting the number of iterations used allows the zone owner to choose the cost of computing a hash, and so the cost of generating a dictionary. Note that this is distinct from the effect of salt, which prevents the use of a single precomputed dictionary for all time.

Obviously the number of iterations also affects the zone owner's cost of signing the zone as well as the verifiers cost of verifying the zone. We therefore impose an upper limit on the number of iterations. We base this on the number of iterations that approximately doubles the cost of signing the zone.

A zone owner MUST NOT use a value higher than shown in the table below for iterations. A resolver MAY treat a response with a higher value as bogus.

RSA Key Size	Iterations
1024	3,000
2048	20,000
4096	150,000

DSA Key Size	Iterations
1024	1,500
2048	5,000

This table is based on 150,000 SHA-1's per second, 50 RSA signs per second for 1024 bit keys, 7 signs per second for 2048 bit keys, 1 sign per second for 4096 bit keys, 100 DSA signs per second for 1024 bit keys and 30 signs per second for 2048 bit keys.

Note that since RSA verifications are 10-100 times faster than signatures (depending on key size), in the case of RSA the legal values of iterations can substantially increase the cost of verification.

8.4. Hash Collision

Hash collisions occur when different messages have the same hash value. The expected number of domain names needed to give a 1 in 2 chance of a single collision is about 2^(n/2) for a hash of length n bits (i.e. 2^80 for SHA-1). Though this probability is extremely low, the following paragraphs deal with avoiding collisions and assessing possible damage in the event of an attack using hash collisions.

8.4.1. Avoiding Hash Collisions during generation

During generation of NSEC3 RRs, hash values are supposedly unique. In the (academic) case of a collision occurring, an alternative salt MUST be chosen and all hash values MUST be regenerated.

8.4.2. Second Preimage Requirement Analysis

A cryptographic hash function has a second-preimage resistance property. The second-preimage resistance property means that it is computationally infeasible to find another message with the same hash value as a given message, i.e. given preimage X, to find a second preimage X' != X such that hash(X) = hash(X'). The work factor for finding a second preimage is of the order of 2^160 for SHA-1. To mount an attack using an existing NSEC3 RR, an adversary needs to find a second preimage.

Assuming an adversary is capable of mounting such an extreme attack, the actual damage is that a response message can be generated which claims that a certain QNAME (i.e. the second pre-image) does exist,

while in reality QNAME does not exist (a false positive), which will

either cause a security aware resolver to re-query for the non-existent name, or to fail the initial query. Note that the adversary can't mount this attack on an existing name but only on a name that the adversary can't choose and does not yet exist.

8.4.3. Possible Hash Value Truncation Method

The previous sections outlined the low probability and low impact of a second-preimage attack. When impact and probability are low, while space in a DNS message is costly, truncation is tempting. Truncation might be considered to allow for shorter ownernames and rdata for hashed labels. In general, if a cryptographic hash is truncated to n bits, then the expected number of domains required to give a 1 in 2 probability of a single collision is approximately 2^(n/2) and the work factor to produce a second preimage is 2^n.

An extreme hash value truncation would be truncating to the shortest possible unique label value. This would be unwise, since the work factor to produce second preimages would then approximate the size of the zone (sketch of proof: if the zone has k entries, then the length of the names when truncated down to uniqueness should be proportional to log_2(k). Since the work factor to produce a second pre-image is 2^n for an n-bit hash, then in this case it is 2^(C log_2(k)) (where C is some constant), i.e. C'k - a work factor of k).

Though the mentioned truncation can be maximized to a certain extreme, the probability of collision increases exponentially for every truncated bit. Given the low impact of hash value collisions and limited space in DNS messages, the balance between truncation profit and collision damage may be determined by local policy. Of course, the size of the corresponding RRSIG RR is not reduced, so truncation is of limited benefit.

Truncation could be signaled simply by reducing the length of the first label in the ownername. Note that there would have to be a corresponding reduction in the length of the Next Hashed Ownername field.

8.4.4. Server Response to a Run-time Collision

In the astronomically unlikely event that a server is unable to prove nonexistence because the hash of the name that does not exist collides with a name that does exist, the server is obviously broken, and should, therefore, return a response with an RCODE of 2 (server failure).

8.4.5. Parameters that Cover the Zone

Secondary servers (and perhaps other entities) need to reliably determine which NSEC3 parameters (that is, hash, salt and iterations) are present at every hashed ownername, in order to be able to choose an appropriate set of NSEC3 records for negative responses. This is indicated by the parameters at the apex: any set of parameters that is used in an NSEC3 record whose original ownername is the apex of the zone MUST be present throughout the zone.

A method to determine which NSEC3 in a complete chain corresponds to the apex is to look for a NSEC3 RRset which has the SOA bit set in the RDATA bit type maps field.

9. Performance Considerations

Iterated hashes impose a performance penalty on both authoritative servers and resolvers. Therefore, the number of iterations should be carefully chosen. In particular it should be noted that a high value for iterations gives an attacker a very good denial of service attack, since the attacker need not bother to verify the results of their queries, and hence has no performance penalty of his own.

On the other hand, nameservers with low query rates and limited bandwidth are already subject to a bandwidth based denial of service attack, since responses are typically an order of magnitude larger than queries, and hence these servers may choose a high value of iterations in order to increase the difficulty of offline attempts to enumerate their namespace without significantly increasing their vulnerability to denial of service attacks.

10. IANA Considerations

IANA needs to allocate a RR type code for NSEC3 from the standard RR type space (type XXX requested). IANA needs to open a new registry for the NSEC3 Hash Functions. The range for this registry is 0-127. Defined types are:

0 is reserved.
1 is SHA-1 ([13]).
127 is experimental.

11. Security Considerations

The NSEC3 records are still susceptible to dictionary attacks (i.e.

the attacker retrieves all the NSEC3 records, then calculates the hashes of all likely domain names, comparing against the hashes found in the NSEC3 records, and thus enumerating the zone). These are substantially more expensive than enumerating the original NSEC records would have been, and in any case, such an attack could also be used directly against the name server itself by performing queries for all likely names, though this would obviously be more detectable. The expense of this off-line attack can be chosen by setting the number of iterations in the NSEC3 RR.

Domains are also susceptible to a precalculated dictionary attack - that is, a list of hashes for all likely names is computed once, then NSEC3 is scanned periodically and compared against the precomputed hashes. This attack is prevented by changing the salt on a regular basis.

Walking the NSEC3 RRs will reveal the total number of records in the zone, and also what types they are. This could be mitigated by adding dummy entries, but certainly an upper limit can always be found.

Hash collisions may occur. If they do, it will be impossible to prove the non-existence of the colliding domain - however, this is fantastically unlikely, and, in any case, DNSSEC already relies on SHA-1 to not collide.

Responses to queries where QNAME equals an NSEC3 ownername that has no other types may be undetectably changed from a NOERROR/NODATA response to a NAME ERROR response.

The Opt-In Flag (O) allows for unsigned names, in the form of delegations to unsigned subzones, to exist within an otherwise signed zone. All unsigned names are, by definition, insecure, and their validity or existence cannot by cryptographically proven.

In general:

Records with unsigned names (whether existing or not) suffer from the same vulnerabilities as records in an unsigned zone. These vulnerabilities are described in more detail in [16] (note in particular sections 2.3, "Name Games" and 2.6, "Authenticated Denial").

Records with signed names have the same security whether or not Opt-In is used.

Note that with or without Opt-In, an insecure delegation may be undetectably altered by an attacker. Because of this, the primary difference in security when using Opt-In is the loss of the ability to prove the existence or nonexistence of an insecure delegation

within the span of an Opt-In NSEC3 record.

In particular, this means that a malicious entity may be able to insert or delete records with unsigned names. These records are normally NS records, but this also includes signed wildcard expansions (while the wildcard record itself is signed, its expanded name is an unsigned name).

For example, if a resolver received the following response from the example zone above:

Example S.1: Response to query for WWW.DOES-NOT-EXIST.EXAMPLE. A

```
RCODE=NOERROR

Answer Section:

Authority Section:
DOES-NOT-EXIST.EXAMPLE. NS     NS.FORGED.
```

```
EXAMPLE.                NSEC   FIRST-SECURE.EXAMPLE. SOA NS \
                               RRSIG DNSKEY
abcd...                 RRSIG  NSEC3 ...

Additional Section:
```

The resolver would have no choice but to accept that the referral to NS.FORGED. is valid. If a wildcard existed that would have been expanded to cover "WWW.DOES-NOT-EXIST.EXAMPLE.", an attacker could have undetectably removed it and replaced it with the forged delegation.

Note that being able to add a delegation is functionally equivalent to being able to add any record type: an attacker merely has to forge a delegation to nameserver under his/her control and place whatever records needed at the subzone apex.

While in particular cases, this issue may not present a significant security problem, in general it should not be lightly dismissed. Therefore, it is strongly RECOMMENDED that Opt-In be used sparingly. In particular, zone signing tools SHOULD NOT default to using Opt-In, and MAY choose to not support Opt-In at all.

12. References

12.1. Normative References

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

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

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

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

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

[6] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

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

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

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

[10] Gudmundsson, O., "Delegation Signer (DS) Resource Record (RR)", RFC 3658, December 2003.

[11] Eastlake, D., Brunner-Williams, E., and B. Manning, "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 2929, September 2000.

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

[13] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

12.2. Informative References

[14] Vixie, P., "Extending DNSSEC-BIS (DNSSEC-TER)", draft-vixie-dnssec-ter-01 (work in progress), June 2004.

[15] Josefsson, Ed., S,., "The Base16, Base32, and Base64 Data Encodings.", draft-josefsson-rfc3548bis-00 (work in progress), October 2005.

[16] Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, August 2004.

Editorial Comments

[Comment.1] Although, strictly speaking, the names *did* exist.

[Comment.2] Note that this method makes it impossible to detect (extremely unlikely) hash collisions.

Appendix A. Example Zone

This is a zone showing its NSEC3 records. They can also be used as test vectors for the hash algorithm.

The data in the example zone is currently broken, as it uses a different base32 alphabet. This shall be fixed in the next release.

```
example.        3600 IN SOA ns1.example. bugs.x.w.example. (
                            1
                            3600
                            300
                            3600000
                            3600 )
                3600 RRSIG  SOA 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            RtctD6aLUU5Md5wOOItilS7JXX1tf58Ql3sK
                            mTXkL13jqLiUFOGg0uzqRh1U9GbydS0P7M0g
                            qYIt90txzE/4+g== )
                3600 NS     ns1.example.
                3600 NS     ns2.example.
                3600 RRSIG  NS 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            hNyyin2JpECIFxW4vsj8RhHcWCQKUXgO+z4l
                            m7g2zM8q3Qpsm/gYIXSF2Rhj6lAG7esR/X9d
                            1SH5r/wfjuCg+g== )
                3600 MX     1 xx.example.
```

```
                3600 RRSIG  MX 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            L/ZDLMSZJKITmSxmM9Kni37/wKQsdSg6FT0l
                            NMm14jy2Stp91Pwp1HQ1hAMkGWAqCMEKPMtU
                            S/o/g5C8VM6ftQ== )
                3600 DNSKEY 257 3 5 (
                            AQOnsGyJvywVjYmiLbh0EwIRuWYcDiB/8blX
                            cpkoxtpe19Oicv6Zko+8brVsTMeMOpcUeGB1
                            zsYKWJ7BvR2894hX
                            ) ; Key ID = 21960
                3600 DNSKEY 256 3 5 (
                            AQO0gEmbZUL6xbD/xQczHbnwYnf+jQjwz/sU
                            5k44rHTt0Ty+3aOdYoome9TjGMhwkkGby1TL
                            ExXT48OGGdbfIme5
                            ) ; Key ID = 62699
                3600 RRSIG  DNSKEY 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            e6EB+K21HbyZzoLUeRDb6+g0+n8XASYe6h+Z
                            xtnB31sQXZgq8MBHeNFDQW9eZw2hjT9zMClx
                            mTkunTYzqWJrmQ== )
                3600 RRSIG  DNSKEY 5 1 3600 20050712112304 (
                            20050612112304 21960 example.
                            SnWLiNWLbOuiKU/F/wVMokvcg6JVzGpQ2VUk
                            ZbKjB9ON0t3cdc+FZbOCMnEHRJiwgqlnncik
                            3w7ZY2UWyYIvpw== )
5pe7ctl7pfs2cilroy5dcofx4rcnlypd.example. 3600 NSEC3  0 1 1 (
                            deadbeaf
                            7nomf47k3vlidh4vxahhpp47l3tgv7a2
                            NSEC3 RRSIG )
                3600 RRSIG  NSEC3 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            PTWYq4WZmmtgh9UQif342HWf9DD9RuuM4ii5
                            Z1oZQgRi5zrsoKHAgl2YXprF2Rfk1TLgsiFQ
                            sb7KfbaUo/vzAg== )
7nomf47k3vlidh4vxahhpp47l3tgv7a2.example. 3600 NSEC3  0 1 1 (
                            deadbeaf
                            dw4o7j64wnel3j4jh7fb3c5n7w3js2yb
                            MX NSEC3 RRSIG )
                3600 RRSIG  NSEC3 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            YTcqole3h8EOsTT3HKnwhR1QS8borR0XtZaA
                            ZrLsx6n0RDC1AAdZONYOvdqvcal9PmwtWjlo
                            MEFQmc/gEuxojA== )
a.example.      3600 IN NS  ns1.a.example.
                3600 IN NS  ns2.a.example.
                3600 DS     58470 5 1 3079F1593EBAD6DC121E202A8B
                            766A6A4837206C )
                3600 RRSIG  DS 5 2 3600 20050712112304 (
```

```
                                20050612112304 62699 example.
                                QavhbsSmEvJLSUzGoTpsV3SKXCpaL1UO3Ehn
                                cB0ObBIlex/Zs9kJyG/9uW1cYYt/1wvgzmX2
                                0kx7rGKTc3RQDA== )
   ns1.a.example. 3600 IN A     192.0.2.5
   ns2.a.example. 3600 IN A     192.0.2.6
   ai.example.    3600 IN A     192.0.2.9
                  3600 RRSIG    A 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                plY5M26ED3Owe3YX0pBIhgg44j89NxUaoBrU
                                6bLRr99HpKfFl1sIy18JiRS7evlxCETZgubq
                                ZXW5S+1VjMZYzQ== )
                  3600 HINFO    "KLH-10" "ITS"
                  3600 RRSIG    HINFO 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                AR0hG/Z/e+vlRhxRQSVIFORzrJTBpdNHhwUk
                                tiuqg+zGqKK84eIqtrqXelcE2szKnF3YPneg
                                VGNmbgPnqDVPiA== )
                  3600 AAAA     2001:db8:0:0:0:0:f00:baa9
                  3600 RRSIG    AAAA 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                PNF/t7+DeosEjhfuL0kmsNJvn16qhYyLI9FV
                                ypSCorFx/PKIlEL3syomkYM2zcXVSRwUXMns
                                l5/UqLCJJ9BDMg== )
   b.example.     3600 IN NS    ns1.b.example.
                  3600 IN NS    ns2.b.example.
   ns1.b.example. 3600 IN A     192.0.2.7
   ns2.b.example. 3600 IN A     192.0.2.8
   dw4o7j64wnel3j4jh7fb3c5n7w3js2yb.example. 3600 NSEC3  0 1 1 (
                                deadbeaf
                                gmnfcccja7wkax3iv26bs75myptje3qk
                                MX DNSKEY NS SOA NSEC3 RRSIG )
                  3600 RRSIG    NSEC3 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                VqEbXiZLJVYmo25fmO3IuHkAX155y8NuA50D
                                C0NmJV/D4R3rLm6tsL6HB3a3f6IBw6kKEa2R
                                MOiKMSHozVebqw== )
   gmnfcccja7wkax3iv26bs75myptje3qk.example. 3600 NSEC3  0 1 1 (
                                deadbeaf
                                jt4bbfokgbmr57qx4nqucvvn7fmo6ab6
                                DS NS NSEC3 RRSIG )
                  3600 RRSIG    NSEC3 5 2 3600 20050712112304 (
```

```
                                20050612112304 62699 example.
                                ZqkdmF6eICpHyn1Cj7Yvw+nLcbji46Qpe76/
                                ZetqdZV7K5sO3ol5dOc0dZyXDqsJp1is5StW
                                OwQBGbOegrW/Zw== )
   jt4bbfokgbmr57qx4nqucvvn7fmo6ab6.example. 3600 NSEC3  0 1 1 (
                                deadbeaf
```



```
                                kcll7fqfnisuhfekckeeqnmbbd4maanu
                                NSEC3 RRSIG )
                   3600 RRSIG   NSEC3 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                FXyCVQUdFF1EW1NcgD2V724/It0rn3lr+30V
                                IyjmqwOMvQ4G599InTpiH46xhX3U/FmUzHOK
                                94Zbq3k8lgdpZA== )
   kcll7fqfnisuhfekckeeqnmbbd4maanu.example. 3600 NSEC3  1 1 1 (
                                deadbeaf
                                n42hbhnjj333xdxeybycax5ufvntux5d
                                MX NSEC3 RRSIG )
                   3600 RRSIG   NSEC3 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                d0g8MTOvVwByOAIwvYV9JrTHwJof1VhnMKuA
                                IBj6Xaeney86RBZYgg7Qyt9WnQSK3uCEeNpx
                                TOLtc5jPrkL4zQ== )
   n42hbhnjj333xdxeybycax5ufvntux5d.example. 3600 NSEC3  0 1 1 (
                                deadbeaf
                                nimwfwcnbeoodmsc6npv3vuaagaevxxu
                                A NSEC3 RRSIG )
                   3600 RRSIG   NSEC3 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                MZGzllh+YFqZbY8SkHxARhXFiMDPS0tvQYyy
                                91tj+lbl45L/BElD3xxB/LZMO8vQejYtMLHj
                                xFPFGRIW3wKnrA== )
   nimwfwcnbeoodmsc6npv3vuaagaevxxu.example. 3600 NSEC3  0 1 1 (
                                deadbeaf
                                vhgwr2qgykdkf4m6iv6vkagbxozphazr
                                HINFO A AAAA NSEC3 RRSIG )
                   3600 RRSIG   NSEC3 5 2 3600 20050712112304 (
                                20050612112304 62699 example.
                                c3zQdK68cYTHTjh1cD6pi0vblXwzyoU/m7Qx
                                z8kaPYikbJ9vgSl9YegjZukgQSwybHUC0SYG
                                jL33Wm1p07TBdw== )
   ns1.example.    3600 A       192.0.2.1
```

```
                3600 RRSIG  A 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            QLGkaqWXxRuE+MHKkMvVlswg65HcyjvD1fyb
                            BDZpcfiMHH9w4x1eRqRamtSDTcqLfUrcYkrr
                            nWWLepz1PjjShQ== )
ns2.example.    3600 A      192.0.2.2
                3600 RRSIG  A 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            UoIZaC1O6XHRWGHBOl8XFQKPdYTkRCz6SYh3
                            P2mZ3xfY22fLBCBDrEnOc8pGDGijJaLl26Cz
                            AkeTJu3J3auUiA== )
vhgwr2qgykdkf4m6iv6vkagbxozphazr.example. 3600 NSEC3  0 1 1 (
                            deadbeaf
```



```
                            wbyijvpnyj33pcpi3i44ecnibnaj7eiw
                            HINFO A AAAA NSEC3 RRSIG )
                3600 RRSIG  NSEC3 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            leFhoF5FXZAiNOxK4OBOOA0WKdbaD5lLDT/W
                            kLoyWnQ6WGBwsUOdsEcVmqz+1n7q9bDf8G8M
                            5SNSHIyfpfsi6A== )
*.w.example.    3600 MX     1 ai.example.
                3600 RRSIG  MX 5 3 3600 20050712112304 (
                            20050612112304 62699 example.
                            sYNUPHn1/gJ87wTHNksGdRm3vfnSFa2BbofF
                            xGfJLF5A4deRu5f0hvxhAFDCcXfIASj7z0wQ
                            gQlgxEwhvQDEaQ== )
x.w.example.    3600 MX     1 xx.example.
                3600 RRSIG  MX 5 3 3600 20050712112304 (
                            20050612112304 62699 example.
                            s1XQ/8SlViiEDik9edYs1Ooe3XiXo453Dg7w
                            lqQoewuDzmtd6RaLNu52W44zTM1EHJES8ujP
                            U9VazOa1KEIq1w== )
x.y.w.example. 3600 MX      1 xx.example.
                3600 RRSIG  MX 5 4 3600 20050712112304 (
                            20050612112304 62699 example.
                            aKVCGO/Fx9rm04UUsHRTTYaDA8o8dGfyq6t7
                            uqAcYxU9xiXP+xNtLHBv7er6Q6f2JbOs6SGF
                            9VrQvJjwbllAfA== )
wbyijvpnyj33pcpi3i44ecnibnaj7eiw.example. 3600 NSEC3  0 1 1 (
                            deadbeaf
                            zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui
```

```
                              A NSEC3 RRSIG )
                 3600 RRSIG   NSEC3 5 2 3600 20050712112304 (
                              20050612112304 62699 example.
                              ledFAaDCqDxapQ1FvBAjjK2DP06iQj8AN6gN
                              ZycTeSmobKLTpzbgQp8uKYYe/DPHjXYmuEhd
                              oorBv4xkb0flXw== )
   xx.example.    3600 A       192.0.2.10
                 3600 RRSIG   A 5 2 3600 20050712112304 (
                              20050612112304 62699 example.
                              XSuMVjNxovbZUsnKU6oQDygaK+WB+O5HYQG9
                              tJgphHIX7TM4uZggfR3pNM+4jeC8nt2OxZZj
                              cxwCXWj82GVGdw== )
                 3600 HINFO   "KLH-10" "TOPS-20"
                 3600 RRSIG   HINFO 5 2 3600 20050712112304 (
                              20050612112304 62699 example.
                              ghS2DimOqPSacG9j6KMgXSfTMSjLxvoxvx3q
                              OKzzPst4tEbAmocF2QX8IrSHr67m4ZLmd2Fk
                              KMf4DgNBDj+dIQ== )
                 3600 AAAA    2001:db8:0:0:0:0:f00:baaa
                 3600 RRSIG   AAAA 5 2 3600 20050712112304 (
```



```
                              20050612112304 62699 example.
                              rto7afZkXYB17IfmQCT5QoEMMrlkeOoAGXzo
                              w8Wmcg86Fc+MQP0hyXFScI1gYNSgSSoDMXIy
                              rzKKwb8J04/ILw== )
   zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui.example. 3600 NSEC3  0 1 1 (
                              deadbeaf
                              5pe7ctl7pfs2cilroy5dcofx4rcnlypd
                              MX NSEC3 RRSIG )
                 3600 RRSIG   NSEC3 5 2 3600 20050712112304 (
                              20050612112304 62699 example.
                              eULkdWjcjmM+wXQcr7zXNfnGLgHjZSJINGkt
                              7Zmvp7WKVAqoHMm1RXV8IfBH1aRgv5+/Lgny
                              OcFlrPGPMm48/A== )
```

Appendix B. Example Responses

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

B.1. answer

A successful query to an authoritative server.


```
;; Header: QR AA DO RCODE=0
;;
;; Question
x.w.example.        IN MX

;; Answer
x.w.example.   3600 IN MX  1 xx.example.
x.w.example.   3600 IN RRSIG  MX 5 3 3600 20050712112304 (
                           20050612112304 62699 example.
                           s1XQ/8SlViiEDik9edYs1Ooe3XiXo453Dg7w
                           lqQoewuDzmtd6RaLNu52W44zTM1EHJES8ujP
                           U9VazOa1KEIq1w== )

;; Authority
```

```
example.        3600 IN NS  ns1.example.
example.        3600 IN NS  ns2.example.
example.        3600 IN RRSIG  NS 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            hNyyin2JpECIFxW4vsj8RhHcWCQKUXgO+z4l
                            m7g2zM8q3Qpsm/gYIXSF2Rhj6lAG7esR/X9d
                            1SH5r/wfjuCg+g== )

;; Additional
xx.example.     3600 IN A   192.0.2.10
xx.example.     3600 IN RRSIG  A 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            XSuMVjNxovbZUsnKU6oQDygaK+WB+O5HYQG9
                            tJgphHIX7TM4uZggfR3pNM+4jeC8nt2OxZZj
                            cxwCXWj82GVGdw== )
xx.example.     3600 IN AAAA   2001:db8::f00:baaa
xx.example.     3600 IN RRSIG  AAAA 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            rto7afZkXYB17IfmQCT5QoEMMrlkeOoAGXzo
                            w8Wmcg86Fc+MQP0hyXFScI1gYNSgSSoDMXIy
                            rzKKwb8J04/ILw== )
ns1.example.    3600 IN A   192.0.2.1
ns1.example.    3600 IN RRSIG  A 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            QLGkaqWXxRuE+MHKkMvVlswg65HcyjvD1fyb
                            BDZpcfiMHH9w4x1eRqRamtSDTcqLfUrcYkrr
                            nWWLepz1PjjShQ== )
ns2.example.    3600 IN A   192.0.2.2
ns2.example.    3600 IN RRSIG  A 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            UoIZaC1O6XHRWGHBOl8XFQKPdYTkRCz6SYh3
                            P2mZ3xfY22fLBCBDrEnOc8pGDGijJaLl26Cz
                            AkeTJu3J3auUiA== )
```


The query returned an MX RRset for "x.w.example". The corresponding RRSIG RR indicates that the MX RRset was signed by an "example" DNSKEY with algorithm 5 and key tag 62699. The resolver needs the corresponding DNSKEY RR in order to authenticate this answer. The discussion below describes how a resolver might obtain this DNSKEY RR.

The RRSIG RR indicates the original TTL of the MX RRset was 3600, and, for the purpose of authentication, the current TTL is replaced by 3600. The RRSIG RR's labels field value of 3 indicates that the answer was not the result of wildcard expansion. The "x.w.example" MX RRset is placed in canonical form, and, assuming the current time falls between the signature inception and expiration dates, the signature is authenticated.

B.1.1. Authenticating the Example DNSKEY RRset

This example shows the logical authentication process that starts from a configured root DNSKEY RRset (or DS RRset) and moves down the tree to authenticate the desired "example" DNSKEY RRset. Note that the logical order is presented for clarity. An implementation may choose to construct the authentication as referrals are received or to construct the authentication chain only after all RRsets have been obtained, or in any other combination it sees fit. The example here demonstrates only the logical process and does not dictate any implementation rules.

We assume the resolver starts with a configured DNSKEY RRset for the root zone (or a configured DS RRset for the root zone). The resolver checks whether this configured DNSKEY RRset is present in the root DNSKEY RRset (or whether a DS RR in the DS RRset matches some DNSKEY RR in the root DNSKEY RRset), whether this DNSKEY RR has signed the root DNSKEY RRset, and whether the signature lifetime is valid. If all these conditions are met, all keys in the DNSKEY RRset are considered authenticated. The resolver then uses one (or more) of the root DNSKEY RRs to authenticate the "example" DS RRset. Note that the resolver may have to query the root zone to obtain the root DNSKEY RRset or "example" DS RRset.

Once the DS RRset has been authenticated using the root DNSKEY, the resolver checks the "example" DNSKEY RRset for some "example" DNSKEY RR that matches one of the authenticated "example" DS RRs. If such a matching "example" DNSKEY is found, the resolver checks whether this DNSKEY RR has signed the "example" DNSKEY RRset and the signature lifetime is valid. If these conditions are met, all keys in the "example" DNSKEY RRset are considered authenticated.

Finally, the resolver checks that some DNSKEY RR in the "example"

DNSKEY RRset uses algorithm 5 and has a key tag of 62699. This DNSKEY is used to authenticate the RRSIG included in the response. If multiple "example" DNSKEY RRs match this algorithm and key tag, then each DNSKEY RR is tried, and the answer is authenticated if any of the matching DNSKEY RRs validate the signature as described above.

B.2. Name Error

An authoritative name error. The NSEC3 RRs prove that the name does not exist and that no covering wildcard exists.

```
;; Header: QR AA DO RCODE=3
;;
;; Question
a.c.x.w.example.          IN A

;; Answer
;; (empty)

;; Authority
example.        3600 IN SOA ns1.example. bugs.x.w.example. (
                            1
                            3600
                            300
                            3600000
                            3600
                            )
example.        3600 IN RRSIG  SOA 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            RtctD6aLUU5Md5wOOItilS7JXX1tf58Ql3sK
                            mTXkL13jqLiUFOGg0uzqRh1U9GbydS0P7M0g
                            qYIt90txzE/4+g== )
7nomf47k3vlidh4vxahhpp47l3tgv7a2.example. 3600 IN NSEC3  0 1 1 (
                            deadbeaf
                            dw4o7j64wnel3j4jh7fb3c5n7w3js2yb
                            MX NSEC3 RRSIG )
7nomf47k3vlidh4vxahhpp47l3tgv7a2.example. 3600 IN RRSIG  NSEC3 (
                            5 2 3600 20050712112304
                            20050612112304 62699 example.
                            YTcqole3h8EOsTT3HKnwhR1QS8borR0XtZaA
                            ZrLsx6n0RDC1AAdZONYOvdqvcal9PmwtWjlo
                            MEFQmc/gEuxojA== )
nimwfwcnbeoodmsc6npv3vuaagaevxxu.example. 3600 IN NSEC3  0 1 1 (
                            deadbeaf
                            vhgwr2qgykdkf4m6iv6vkagbxozphazr
                            HINFO A AAAA NSEC3 RRSIG )
nimwfwcnbeoodmsc6npv3vuaagaevxxu.example. 3600 IN RRSIG  NSEC3 (
                            5 2 3600 20050712112304
                            20050612112304 62699 example.
                            c3zQdK68cYTHTjh1cD6pi0vblXwzyoU/m7Qx
                            z8kaPYikbJ9vgSl9YegjZukgQSwybHUC0SYG
                            jL33Wm1p07TBdw== )
;; Additional
;; (empty)
```

The query returned two NSEC3 RRs that prove that the requested data does not exist and no wildcard applies. The negative reply is authenticated by verifying both NSEC3 RRs. The NSEC3 RRs are authenticated in a manner identical to that of the MX RRset discussed

above. At least one of the owner names of the NSEC3 RRs will match the closest encloser. At least one of the NSEC3 RRs prove that there exists no longer name. At least one of the NSEC3 RRs prove that there exists no wildcard RRsets that should have been expanded. The closest encloser can be found by hashing the apex ownername (The SOA RR's ownername, or the ownername of the DNSKEY RRset referred by an RRSIG RR), matching it to the ownername of one of the NSEC3 RRs, and if that fails, continue by adding labels. In other words, the resolver first hashes example, checks for a matching NSEC3 ownername, then hashes w.example, checks, and finally hashes w.x.example and checks.

In the above example, the name 'x.w.example' hashes to '7nomf47k3vlidh4vxahhpp47l3tgv7a2'. This indicates that this might be the closest encloser. To prove that 'c.x.w.example' and '*.x.w.example' do not exists, these names are hashed to respectively 'qsgoxsf2lanysajhtmaylde4tqwnqppl' and 'cvljzyf6nsckjowghch4tt3nohocpdka'. The two NSEC3 records prove that these hashed ownernames do not exists, since the names are within the given intervals.

B.3. No Data Error

A "no data" response. The NSEC3 RR proves that the name exists and that the requested RR type does not.

```
;; Header: QR AA DO RCODE=0
;;
;; Question
ns1.example.        IN MX

;; Answer
;; (empty)

;; Authority
example.       3600 IN SOA ns1.example. bugs.x.w.example. (
                           1
                           3600
                           300
                           3600000
                           3600
                           )
example.       3600 IN RRSIG  SOA 5 1 3600 20050712112304 (
                           20050612112304 62699 example.
                           RtctD6aLUU5Md5wOOItilS7JXX1tf58Ql3sK
                           mTXkL13jqLiUFOGg0uzqRh1U9GbydS0P7M0g
                           qYIt90txzE/4+g== )
wbyijvpnyj33pcpi3i44ecnibnaj7eiw.example. 3600 IN NSEC3  0 1 1 (
                           deadbeaf
                           zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui
                           A NSEC3 RRSIG )
wbyijvpnyj33pcpi3i44ecnibnaj7eiw.example. 3600 IN RRSIG  NSEC3 (
                           5 2 3600 20050712112304
                           20050612112304 62699 example.
                           ledFAaDCqDxapQ1FvBAjjK2DP06iQj8AN6gN
```

```
                                ZycTeSmobKLTpzbgQp8uKYYe/DPHjXYmuEhd
                                oorBv4xkb0flXw== )
    ;; Additional
    ;; (empty)
```

The query returned an NSEC3 RR that proves that the requested name exists ("ns1.example." hashes to "wbyijvpnyj33pcpi3i44ecnibnaj7eiw"), but the requested RR type does not exist (type MX is absent in the type code list of the NSEC RR). The negative reply is authenticated by verifying the NSEC3 RR. The NSEC3 RR is authenticated in a manner identical to that of the MX RRset discussed above.

B.3.1. No Data Error, Empty Non-Terminal

A "no data" response because of an empty non-terminal. The NSEC3 RR proves that the name exists and that the requested RR type does not.


```
    ;; Header: QR AA DO RCODE=0
    ;;
    ;; Question
    y.w.example.        IN A

    ;; Answer
    ;; (empty)

    ;; Authority
    example.       3600 IN SOA ns1.example. bugs.x.w.example. (
                               1
                               3600
                               300
                               3600000
                               3600
                               )
    example.       3600 IN RRSIG  SOA 5 1 3600 20050712112304 (
                               20050612112304 62699 example.
                               RtctD6aLUU5Md5wOOItilS7JXX1tf58Ql3sK
                               mTXkL13jqLiUFOGg0uzqRh1U9GbydS0P7M0g
                               qYIt90txzE/4+g== )
    jt4bbfokgbmr57qx4nqucvvn7fmo6ab6.example. 3600 IN NSEC3  0 1 1 (
```

```
                              deadbeaf
                              kcll7fqfnisuhfekckeeqnmbbd4maanu
                              NSEC3 RRSIG )
jt4bbfokgbmr57qx4nqucvvn7fmo6ab6.example. 3600 IN RRSIG  NSEC3 (
                              5 2 3600 20050712112304
                              20050612112304 62699 example.
                              FXyCVQUdFF1EW1NcgD2V724/It0rn3lr+30V
                              IyjmqwOMvQ4G599InTpiH46xhX3U/FmUzHOK
                              94Zbq3k8lgdpZA== )
```

The query returned an NSEC3 RR that proves that the requested name exists ("y.w.example." hashes to "jt4bbfokgbmr57qx4nqucvvn7fmo6ab6"), but the requested RR type does not exist (Type A is absent in the type-bit-maps of the NSEC3 RR). The negative reply is authenticated by verifying the NSEC3 RR. The NSEC3 RR is authenticated in a manner identical to that of the MX RRset discussed above. Note that, unlike generic empty non terminal proof using NSECs, this is identical to proving a No Data Error. This example is solely mentioned to be complete.

B.4. Referral to Signed Zone

Referral to a signed zone. The DS RR contains the data which the resolver will need to validate the corresponding DNSKEY RR in the child zone's apex.


```
;; Header: QR DO RCODE=0
;;

;; Question
mc.a.example.       IN MX

;; Answer
;; (empty)

;; Authority
a.example.     3600 IN NS  ns1.a.example.
a.example.     3600 IN NS  ns2.a.example.
a.example.     3600 IN DS  58470 5 1 (
                           3079F1593EBAD6DC121E202A8B766A6A4837
                           206C )
```

```
a.example.     3600 IN RRSIG  DS 5 2 3600 20050712112304 (
                              20050612112304 62699 example.
                              QavhbsSmEvJLSUzGoTpsV3SKXCpaL1UO3Ehn
                              cB0ObBIlex/Zs9kJyG/9uW1cYYt/1wvgzmX2
                              0kx7rGKTc3RQDA== )

;; Additional
ns1.a.example. 3600 IN A   192.0.2.5
ns2.a.example. 3600 IN A   192.0.2.6
```

The query returned a referral to the signed "a.example." zone. The DS RR is authenticated in a manner identical to that of the MX RRset discussed above. This DS RR is used to authenticate the "a.example" DNSKEY RRset.

Once the "a.example" DS RRset has been authenticated using the "example" DNSKEY, the resolver checks the "a.example" DNSKEY RRset for some "a.example" DNSKEY RR that matches the DS RR. If such a matching "a.example" DNSKEY is found, the resolver checks whether this DNSKEY RR has signed the "a.example" DNSKEY RRset and whether the signature lifetime is valid. If all these conditions are met, all keys in the "a.example" DNSKEY RRset are considered authenticated.

B.5. Referral to Unsigned Zone using the Opt-In Flag

The NSEC3 RR proves that nothing for this delegation was signed in the parent zone. There is no proof that the delegation exists


```
;; Header: QR DO RCODE=0
;;
;; Question
mc.b.example.       IN MX

;; Answer
;; (empty)
```

```
;; Authority
b.example.     3600 IN NS  ns1.b.example.
b.example.     3600 IN NS  ns2.b.example.
kcll7fqfnisuhfekckeeqnmbbd4maanu.example. 3600 IN NSEC3  1 1 1 (
                            deadbeaf
                            n42hbhnjj333xdxeybycax5ufvntux5d
                            MX NSEC3 RRSIG )
kcll7fqfnisuhfekckeeqnmbbd4maanu.example. 3600 IN RRSIG  NSEC3 (
                            5 2 3600 20050712112304
                            20050612112304 62699 example.
                            d0g8MTOvVwByOAIwvYV9JrTHwJof1VhnMKuA
                            IBj6Xaeney86RBZYgg7Qyt9WnQSK3uCEeNpx
                            TOLtc5jPrkL4zQ== )

;; Additional
ns1.b.example. 3600 IN A   192.0.2.7
ns2.b.example. 3600 IN A   192.0.2.8
```

The query returned a referral to the unsigned "b.example." zone. The NSEC3 proves that no authentication leads from "example" to "b.example", since the hash of "b.example" ("ldjpfcucebeks5azmzpty4qlel4cftzo") is within the NSEC3 interval and the NSEC3 opt-in bit is set. The NSEC3 RR is authenticated in a manner identical to that of the MX RRset discussed above.

B.6. Wildcard Expansion

A successful query that was answered via wildcard expansion. The label count in the answer's RRSIG RR indicates that a wildcard RRset was expanded to produce this response, and the NSEC3 RR proves that no closer match exists in the zone.


```
;; Header: QR AA DO RCODE=0
```

```
;;
;; Question
a.z.w.example.      IN MX

;; Answer
a.z.w.example. 3600 IN MX  1 ai.example.
a.z.w.example. 3600 IN RRSIG  MX 5 3 3600 20050712112304 (
                            20050612112304 62699 example.
                            sYNUPHn1/gJ87wTHNksGdRm3vfnSFa2BbofF
                            xGfJLF5A4deRu5f0hvxhAFDCcXfIASj7z0wQ
                            gQlgxEwhvQDEaQ== )
;; Authority
example.       3600 NS     ns1.example.
example.       3600 NS     ns2.example.
example.       3600 IN RRSIG  NS 5 1 3600 20050712112304 (
                            20050612112304 62699 example.
                            hNyyin2JpECIFxW4vsj8RhHcWCQKUXgO+z4l
                            m7g2zM8q3Qpsm/gYIXSF2Rhj6lAG7esR/X9d
                            1SH5r/wfjuCg+g== )
zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui.example. 3600 IN NSEC3  0 1 1 (
                            deadbeaf
                            5pe7ctl7pfs2cilroy5dcofx4rcnlypd
                            MX NSEC3 RRSIG )
zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui.example. 3600 IN RRSIG  NSEC3 (
                            5 2 3600 20050712112304
                            20050612112304 62699 example.
                            eULkdWjcjmM+wXQcr7zXNfnGLgHjZSJINGkt
                            7Zmvp7WKVAqoHMm1RXV8IfBH1aRgv5+/Lgny
                            OcFlrPGPMm48/A== )
;; Additional
ai.example.    3600 IN A   192.0.2.9
ai.example.    3600 IN RRSIG  A 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            plY5M26ED3Owe3YX0pBIhgg44j89NxUaoBrU
                            6bLRr99HpKfFl1sIy18JiRS7evlxCETZgubq
                            ZXW5S+1VjMZYzQ== )
ai.example.    3600 AAAA   2001:db8::f00:baa9
ai.example.    3600 IN RRSIG  AAAA 5 2 3600 20050712112304 (
                            20050612112304 62699 example.
                            PNF/t7+DeosEjhfuL0kmsNJvn16qhYyLI9FV
                            ypSCorFx/PKIlEL3syomkYM2zcXVSRwUXMns
                            l5/UqLCJJ9BDMg== )
```

The query returned an answer that was produced as a result of wildcard expansion. The answer section contains a wildcard RRset expanded as it would be in a traditional DNS response, and the corresponding RRSIG indicates that the expanded wildcard MX RRset was

signed by an "example" DNSKEY with algorithm 5 and key tag 62699. The RRSIG indicates that the original TTL of the MX RRset was 3600, and, for the purpose of authentication, the current TTL is replaced by 3600. The RRSIG labels field value of 2 indicates that the answer is the result of wildcard expansion, as the "a.z.w.example" name contains 4 labels. The name "a.z.w.example" is replaced by "*.w.example", the MX RRset is placed in canonical form, and, assuming that the current time falls between the signature inception and expiration dates, the signature is authenticated.

The NSEC3 proves that no closer match (exact or closer wildcard) could have been used to answer this query, and the NSEC3 RR must also be authenticated before the answer is considered valid.

B.7. Wildcard No Data Error

A "no data" response for a name covered by a wildcard. The NSEC3 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 DO RCODE=0
;;
;; Question
a.z.w.example.      IN AAAA

;; Answer
;; (empty)

;; Authority
example.       3600 IN SOA ns1.example. bugs.x.w.example. (
                           1
                           3600
                           300
                           3600000
                           3600
                           )
example.       3600 IN RRSIG  SOA 5 1 3600 20050712112304 (
                           20050612112304 62699 example.
                           RtctD6aLUU5Md5wOOItilS7JXX1tf58Ql3sK
                           mTXkL13jqLiUFOGg0uzqRh1U9GbydS0P7M0g
                           qYIt90txzE/4+g== )
zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui.example. 3600 IN NSEC3  0 1 1 (
                           deadbeaf
                           5pe7ctl7pfs2cilroy5dcofx4rcnlypd
                           MX NSEC3 RRSIG )
zjxfz5o7t4ty4u3f6fa7mhhqzjln4mui.example. 3600 IN RRSIG  NSEC3 (
                           5 2 3600 20050712112304
                           20050612112304 62699 example.
                           eULkdWjcjmM+wXQcr7zXNfnGLgHjZSJINGkt
                           7Zmvp7WKVAqoHMm1RXV8IfBH1aRgv5+/Lgny
                           OcFlrPGPMm48/A== )
;; Additional
;; (empty)
```

The query returned NSEC3 RRs that prove that the requested data does not exist and no wildcard applies. The negative reply is authenticated by verifying both NSEC3 RRs.

B.8. DS Child Zone No Data Error

A "no data" response for a QTYPE=DS query that was mistakenly sent to a name server for the child zone.


```
;; Header: QR AA DO RCODE=0
;;
;; Question
example.            IN DS

;; Answer
;; (empty)

;; Authority
example.       3600 IN SOA ns1.example. bugs.x.w.example. (
                           1
                           3600
                           300
                           3600000
                           3600
                           )
example.       3600 IN RRSIG  SOA 5 1 3600 20050712112304 (
                           20050612112304 62699 example.
                           RtctD6aLUU5Md5wOOItilS7JXX1tf58Ql3sK
                           mTXkL13jqLiUFOGg0uzqRh1U9GbydS0P7M0g
                           qYIt90txzE/4+g== )
dw4o7j64wnel3j4jh7fb3c5n7w3js2yb.example. 3600 IN NSEC3  0 1 1 (
                           deadbeaf
                           gmnfcccja7wkax3iv26bs75myptje3qk
                           MX DNSKEY NS SOA NSEC3 RRSIG )
dw4o7j64wnel3j4jh7fb3c5n7w3js2yb.example. 3600 IN RRSIG  NSEC3 (
                           5 2 3600 20050712112304
                           20050612112304 62699 example.
                           VqEbXiZLJVYmo25fmO3IuHkAX155y8NuA50D
                           C0NmJV/D4R3rLm6tsL6HB3a3f6IBw6kKEa2R
                           MOiKMSHozVebqw== )
```

```
;; Additional
;; (empty)
```

The query returned NSEC RRs that shows the requested was answered by a child server ("example" server). The NSEC RR indicates the presence of an SOA RR, showing that the answer is from the child . Queries for the "example" DS RRset should be sent to the parent servers ("root" servers).

Authors' Addresses

Ben Laurie
Nominet
17 Perryn Road
London W3 7LR
England

Phone: +44 (20) 8735 0686
Email: ben@algroup.co.uk

Geoffrey Sisson
Nominet

Roy Arends
Nominet

Intellectual Property Statement

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.