

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: September 12, 2019

F. Gont
SI6 Networks / UTN-FRH
I. Arce
Quarkslab
March 11, 2019

Security and Privacy Implications of Numeric Identifiers Employed in
Network Protocols
draft-gont-predictable-numeric-ids-03

Abstract

This document performs an analysis of the security and privacy implications of different types of "numeric identifiers" used in IETF protocols, and tries to categorize them based on their interoperability requirements and the associated failure severity when such requirements are not met. It describes a number of algorithms that have been employed in real implementations to meet such requirements and analyzes their security and privacy properties. Additionally, it provides advice on possible algorithms that could be employed to satisfy the interoperability requirements of each identifier type, while minimizing the security and privacy implications, thus providing guidance to protocol designers and protocol implementers. Finally, it provides recommendations for future protocol specifications regarding the specification of the aforementioned numeric identifiers.

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 <https://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 September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Threat Model	5
4. Issues with the Specification of Identifiers	5
5. Timeline of Vulnerability Disclosures Related to Some Sample Identifiers	6
5.1. IPv4/IPv6 Identification	6
5.2. TCP Initial Sequence Numbers (ISNs)	7
6. Protocol Failure Severity	9
7. Categorizing Identifiers	9
8. Common Algorithms for Identifier Generation	11
8.1. Category #1: Uniqueness (soft failure)	11
8.1.1. Simple Randomization Algorithm	11
8.1.2. Another Simple Randomization Algorithm	12
8.2. Category #2: Uniqueness (hard failure)	13
8.3. Category #3: Uniqueness, constant within context (soft-failure)	13
8.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)	14
8.4.1. Predictable Linear Identifiers Algorithm	14
8.4.2. Per-context Counter Algorithm	16
8.4.3. Simple Hash-Based Algorithm	18
8.4.4. Double-Hash Algorithm	20
8.4.5. Random-Increments Algorithm	21
9. Common Vulnerabilities Associated with Identifiers	23
9.1. Category #1: Uniqueness (soft failure)	23
9.2. Category #2: Uniqueness (hard failure)	23
9.3. Category #3: Uniqueness, constant within context (soft	

failure)	23
9.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)	24
10. Security and Privacy Requirements for Identifiers	25
11. IANA Considerations	26
12. Security Considerations	26
13. Acknowledgements	26
14. References	26
14.1. Normative References	26
14.2. Informative References	27
Authors' Addresses	31

1. Introduction

Network protocols employ a variety of numeric identifiers for different protocol entities, ranging from DNS Transaction IDs (TxIDs) to transport protocol numbers (e.g. TCP ports) or IPv6 Interface Identifiers (IIDs). These identifiers usually have specific properties that must be satisfied such that they do not result in negative interoperability implications (e.g. uniqueness during a specified period of time), and associated failure severities when such properties are not met, ranging from soft to hard failures.

For more than 30 years, a large number of implementations of the TCP/IP protocol suite have been subject to a variety of attacks, with effects ranging from Denial of Service (DoS) or data injection, to information leakage that could be exploited for pervasive monitoring [RFC7528]. The root of these issues has been, in many cases, the poor selection of identifiers in such protocols, usually as a result of an insufficient or misleading specification. While it is generally trivial to identify an algorithm that can satisfy the interoperability requirements for a given identifier, there exists practical evidence that doing so without negatively affecting the security and/or privacy properties of the aforementioned protocols is prone to error.

For example, implementations have been subject to security and/or privacy issues resulting from:

- o Predictable TCP sequence numbers
- o Predictable transport protocol numbers
- o Predictable IPv4 or IPv6 Fragment Identifiers
- o Predictable IPv6 IIDs
- o Predictable DNS TxIDs

Recent history indicates that when new protocols are standardized or new protocol implementations are produced, the security and privacy properties of the associated identifiers tend to be overlooked and inappropriate algorithms to generate identifier values are either suggested in the specification or selected by implementators. As a result, we believe that advice in this area is warranted.

This document contains a non-exhaustive survey of identifiers employed in various IETF protocols, and aims to categorize such identifiers based on their interoperability requirements, and the associated failure severity when such requirements are not met. Subsequently, it analyzes several algorithms that have been employed in real implementation to meet such requirements and analyzes their security and privacy properties, and provides advice on possible algorithms that could be employed to satisfy the interoperability requirements of each category, while minimizing the associated security and privacy implications. Finally, it provides recommendations for future protocol specifications regarding the specification of the aforementioned numeric identifiers.

2. Terminology

Identifier:

A data object in a protocol specification that can be used to definitely distinguish a protocol object (a datagram, network interface, transport protocol endpoint, session, etc) from all other objects of the same type, in a given context. Identifiers are usually defined as a series of bits and represented using integer values. We note that different identifiers may have additional requirements or properties depending on their specific use in a protocol. We use the term "identifier" as a generic term to refer to any data object in a protocol specification that satisfies the identification property stated above.

Failure Severity:

The consequences of a failure to comply with the interoperability requirements of a given identifier. Severity considers the worst potential consequence of a failure, determined by the system damage and/or time lost to repair the failure. In this document we define two types of failure severity: "soft" and "hard".

Hard Failure:

A hard failure is a non-recoverable condition in which a protocol does not operate in the prescribed manner or it operates with excessive degradation of service. For example, an established TCP connection that is aborted due to an error condition constitutes, from the point of view of the transport protocol, a hard failure,

since it enters a state from which normal operation cannot be recovered.

Soft Failure:

A soft failure is a recoverable condition in which a protocol does not operate in the prescribed manner but normal operation can be resumed automatically in a short period of time. For example, a simple packet-loss event that is subsequently recovered with a retransmission can be considered a soft failure.

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 [RFC2119].

3. Threat Model

Throughout this document, we assume an attacker does not have physical or logical device to the device(s) being attacked. We assume the attacker can simply send any traffic to the target devices, to e.g. sample identifiers employed by such devices.

4. Issues with the Specification of Identifiers

While assessing protocol specifications regarding the use of identifiers, we found that most of the issues discussed in this document arise as a result of one of the following:

- o Protocol specifications which under-specify the requirements for their identifiers
- o Protocol specifications that over-specify their identifiers
- o Protocol implementations that simply fail to comply with the specified requirements

A number of protocol implementations (too many of them) simply overlook the security and privacy implications of identifiers. Examples of them are the specification of TCP port numbers in [RFC0793], the specification of TCP sequence numbers in [RFC0793], or the specification of the DNS TxID in [RFC1035].

On the other hand, there are a number of protocol specifications that over-specify some of their associated protocol identifiers. For example, [RFC4291] essentially results in link-layer addresses being embedded in the IPv6 Interface Identifiers (IIDs) when the interoperability requirement of uniqueness could be achieved in other ways that do not result in negative security and privacy implications [RFC7721]. Similarly, [RFC2460] suggests the use of a global counter

for the generation of Fragment Identification values, when the interoperability properties of uniqueness per {Src IP, Dst IP} could be achieved with other algorithms that do not result in negative security and privacy implications.

Finally, there are protocol implementations that simply fail to comply with existing protocol specifications. For example, some popular operating systems (notably Microsoft Windows) still fail to implement randomization of transport protocol ephemeral ports, as specified in [RFC6056].

5. Timeline of Vulnerability Disclosures Related to Some Sample Identifiers

This section contains a non-exhaustive timeline of vulnerability disclosures related to some sample identifiers and other work that has led to advances in this area. The goal of this timeline is to illustrate:

- o That vulnerabilities related to how the values for some identifiers are generated and assigned have affected implementations for an extremely long period of time.
- o That such vulnerabilities, even when addressed for a given protocol version, were later reintroduced in new versions or new implementations of the same protocol.
- o That standardization efforts that discuss and provide advice in this area can have a positive effect on protocol specifications and protocol implementations.

5.1. IPv4/IPv6 Identification

December 1998:

[Sanfilippo1998a] finds that predictable IPv4 Identification values can be leveraged to count the number of packets sent by a target node. [Sanfilippo1998b] explains how to leverage the same vulnerability to implement a port-scanning technique known as dumb/idle scan. A tool that implements this attack is publicly released.

November 1999:

[Sanfilippo1999] discusses how to leverage predictable IPv4 Identification to uncover the rules of a number of firewalls.

November 1999:

[Bellovin2002] explains how the IPv4 Identification field can be exploited to count the number of systems behind a NAT.

December 2003:

[Zalewski2003] explains a technique to perform TCP data injection attack based on predictable IPv4 identification values which requires less effort than TCP injection attacks performed with bare TCP packets.

November 2005:

[Silbersack2005] discusses shortcoming in a number of techniques to mitigate predictable IPv4 Identification values.

October 2007:

[Klein2007] describes a weakness in the pseudo random number generator (PRNG) in use for the generation of the IP Identification by a number of operating systems.

June 2011:

[Gont2011] describes how to perform idle scan attacks in IPv6.

November 2011:

Linux mitigates predictable IPv6 Identification values
[RedHat2011] [SUSE2011] [Ubuntu2011].

December 2011:

[I-D.ietf-6man-predictable-fragment-id-08] describes the security implications of predictable IPv6 Identification values, and possible mitigations.

May 2012:

[Gont2012] notes that some major IPv6 implementations still employ predictable IPv6 Identification values.

June 2015:

[I-D.ietf-6man-predictable-fragment-id-08] notes that some popular host and router implementations still employ predictable IPv6 Identification values.

5.2. TCP Initial Sequence Numbers (ISNs)

September 1981:

[RFC0793], suggests the use of a global 32-bit ISN generator, whose lower bit is incremented roughly every 4 microseconds. However, such an ISN generator makes it trivial to predict the ISN that a TCP will use for new connections, thus allowing a variety of attacks against TCP.

February 1985:

[Morris1985] was the first to describe how to exploit predictable TCP ISNs for forging TCP connections that could then be leveraged for trust relationship exploitation.

April 1989:

[Bellovin1989] discussed the security implications of predictable ISNs (along with a range of other protocol-based vulnerabilities).

February 1995:

[Shimomura1995] reported a real-world exploitation of the attack described in 1985 (ten years before) in [Morris1985].

May 1996:

[RFC1948] was the first IETF effort, authored by Steven Bellovin, to address predictable TCP ISNs. The same concept specified in this document for TCP ISNs was later proposed for TCP ephemeral ports [RFC6056], TCP Timestamps, and eventually even IPv6 Interface Identifiers [RFC7217].

March 2001:

[Zalewski2001] provides a detailed analysis of statistical weaknesses in some ISN generators, and includes a survey of the algorithms in use by popular TCP implementations.

May 2001:

Vulnerability advisories [CERT2001] [USCERT2001] are released regarding statistical weaknesses in some ISN generators, affecting popular TCP/IP implementations.

March 2002:

[Zalewski2002] updates and complements [Zalewski2001]. It concludes that "while some vendors [...] reacted promptly and tested their solutions properly, many still either ignored the issue and never evaluated their implementations, or implemented a flawed solution that apparently was not tested using a known approach". [Zalewski2002].

February 2012:

[RFC6528], after 27 years of Morris' original work [Morris1985], formally updates [RFC0793] to mitigate predictable TCP ISNs.

August 2014:

[I-D.eddy-rfc793bis-04], the upcoming revision of the core TCP protocol specification, incorporates the algorithm specified in [RFC6528] as the recommended algorithm for TCP ISN generation.

6. Protocol Failure Severity

Section 2 defines the concept of "Failure Severity" and two types of failures that we employ throughout this document: soft and hard.

Our analysis of the severity of a failure is performed from the point of view of the protocol in question. However, the corresponding severity on the upper application or protocol may not be the same as that of the protocol in question. For example, a TCP connection that is aborted may or may not result in a hard failure of the upper application: if the upper application can establish a new TCP connection without any impact on the application, a hard failure at the TCP protocol may have no severity at the application level. On the other hand, if a hard failure of a TCP connection results in excessive degradation of service at the application layer, it will also result in a hard failure at the application.

7. Categorizing Identifiers

This section includes a non-exhaustive survey of identifiers, and proposes a number of categories that can accommodate these identifiers based on their interoperability requirements and their failure modes (soft or hard)

Identifier	Interoperability Requirements	Failure Severity
IPv6 Frag ID	Uniqueness (for IP address pair)	Soft/Hard (1)
IPv6 IID	Uniqueness (and constant within IPv6 prefix) (2)	Soft (3)
TCP SEQ	Monotonically-increasing	Hard (4)
TCP eph. port	Uniqueness (for connection ID)	Hard
IPv6 Flow L.	Uniqueness	None (5)
DNS TxID	Uniqueness	None (6)

Table 1: Survey of Identifiers

Notes:

- (1)
While a single collision of Fragment ID values would simply lead to a single packet drop (and hence a "soft" failure), repeated collisions at high data rates might trash the Fragment ID space, leading to a hard failure [RFC4963].
- (2)
While the interoperability requirements are simply that the Interface ID results in a unique IPv6 address, for operational reasons it is typically desirable that the resulting IPv6 address (and hence the corresponding Interface ID) be constant within each network [I-D.ietf-6man-default-iids] [RFC7217].
- (3)
While IPv6 Interface IDs must result in unique IPv6 addresses, IPv6 Duplicate Address Detection (DAD) [RFC4862] allows for the detection of duplicate Interface IDs/addresses, and hence such Interface ID collisions can be recovered.
- (4)
In theory there are no interoperability requirements for TCP sequence numbers, since the TIME-WAIT state and TCP's "quiet time" take care of old segments from previous incarnations of the connection. However, a widespread optimization allows for a new incarnation of a previous connection to be created if the Initial Sequence Number (ISN) of the incoming SYN is larger than the last sequence number seen in that direction for the previous incarnation of the connection. Thus, monotonically-increasing TCP sequence numbers allow for such optimization to work as expected [RFC6528].
- (5)
The IPv6 Flow Label is typically employed for load sharing [RFC7098], along with the Source and Destination IPv6 addresses. Reuse of a Flow Label value for the same set {Source Address, Destination Address} would typically cause both flows to be multiplexed into the same link. However, as long as this does not occur deterministically, it will not result in any negative implications.
- (6)
DNS TxIDs are employed, together with the Source Address, Destination Address, Source Port, and Destination Port, to match DNS requests and responses. However, since an implementation knows which DNS requests were sent for that set of {Source Address, Destination Address, Source Port, and Destination Port, DNS TxID}, a collision of TxID would result, if anything, in a small performance penalty (the response would be discarded when it

is found that it does not answer the query sent in the corresponding DNS query).

Based on the survey above, we can categorize identifiers as follows:

Cat #	Category	Sample Proto IDs
1	Uniqueness (soft failure)	IPv6 Flow L., DNS TxIDs
2	Uniqueness (hard failure)	IPv6 Frag ID, TCP ephemeral port
3	Uniqueness, constant within context (soft failure)	IPv6 IIDs
4	Uniqueness, monotonically increasing within context (hard failure)	TCP ISN

Table 2: Identifier Categories

We note that Category #4 could be considered a generalized case of category #3, in which a monotonically increasing element is added to a constant (within context) element, such that the resulting identifiers are monotonically increasing within a specified context. That is, the same algorithm could be employed for both #3 and #4, given appropriate parameters.

8. Common Algorithms for Identifier Generation

The following subsections describe common algorithms found for Protocol ID generation for each of the categories above.

8.1. Category #1: Uniqueness (soft failure)

8.1.1. Simple Randomization Algorithm

```
/* Ephemeral port selection function */
id_range = max_id - min_id + 1;
next_id = min_id + (random() % id_range);
count = next_id;

do {
    if(check_suitable_id(next_id))
        return next_id;

    if (next_id == max_id) {
        next_id = min_id;
    } else {
        next_id++;
    }

    count--;
} while (count > 0);

return ERROR;
```

Note:

random() is a function that returns a pseudo-random unsigned integer number of appropriate size. Note that the output needs to be unpredictable, and typical implementations of POSIX random() function do not necessarily meet this requirement. See [RFC4086] for randomness requirements for security.

The function check_suitable_id() can check, when possible, whether this identifier is e.g. already in use. When already used, this algorithm selects the next available protocol ID.

All the variables (in this and all the algorithms discussed in this document) are unsigned integers.

8.1.2. Another Simple Randomization Algorithm

The following pseudo-code illustrates another algorithm for selecting a random identifier in which, in the event the identifier is found to be not suitable (e.g., already in use), another identifier is selected randomly:

```
id_range = max_id - min_id + 1;
next_id = min_id + (random() % id_range);
count = id_range;

do {
    if(check_suitable_id(next_id))
        return next_id;

    next_id = min_id + (random() % id_range);
    count--;
} while (count > 0);

return ERROR;
```

This algorithm might be unable to select an identifier (i.e., return "ERROR") even if there are suitable identifiers available, when there are a large number of identifiers "in use".

8.2. Category #2: Uniqueness (hard failure)

One of the most trivial approaches for achieving uniqueness for an identifier (with a hard failure mode) is to implement a linear function. As a result, all of the algorithms described in Section 8.4 are of use for complying the requirements of this identifier category.

8.3. Category #3: Uniqueness, constant within context (soft-failure)

The goal of this algorithm is to produce identifiers that are constant for a given context, but that change when the aforementioned context changes.

Keeping one value for each possible "context" may in many cases be considered too onerous in terms of memory requirements. As a workaround, the following algorithm employs a calculated technique (as opposed to keeping state in memory) to maintain the constant identifier for each given context.

In the following algorithm, the function F() provides (statelessly) a constant identifier for each given context.

```
/* Protocol ID selection function */
id_range = max_id - min_id + 1;

counter = 0;

do {
    offset = F(CONTEXT, counter, secret_key);
    next_id = min_id + (offset % id_range);

    if(check_suitable_id(next_id))
        return next_id;

    counter++;
} while (counter <= MAX_RETRIES);

return ERROR;
```

The function `F()` provides a "per-CONTEXT" constant identifier for a given context. 'offset' may take any value within the storage type range since we are restricting the resulting identifier to be in the range `[min_id, max_id]` in a similar way as in the algorithm described in Section 8.1.1. Collisions can be recovered by incrementing the 'counter' variable and recomputing `F()`.

The function `F()` should be a cryptographic hash function like SHA-256 [FIPS-SHS]. Note: MD5 [RFC1321] is considered unacceptable for `F()` [RFC6151]. CONTEXT is the concatenation of all the elements that define a given context. For example, if this algorithm is expected to produce identifiers that are unique per network interface card (NIC) and SLAAC autoconfiguration prefix, the CONTEXT should be the concatenation of e.g. the interface index and the SLAAC autoconfiguration prefix (please see [RFC7217] for an implementation of this algorithm for the generation of IPv6 IIDs).

The secret should be chosen to be as random as possible (see [RFC4086] for recommendations on choosing secrets).

8.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)

8.4.1. Predictable Linear Identifiers Algorithm

One of the most trivial ways to achieve uniqueness with a low identifier reuse frequency is to produce a linear sequence. This obviously assumes that each identifier will be used for a similar period of time.

For example, the following algorithm has been employed in a number of operating systems for selecting IP fragment IDs, TCP ephemeral ports, etc.

```
/* Initialization at system boot time. Could be random */
next_id = min_id;
id_inc= 1;

/* Identifier selection function */
count = max_id - min_id + 1;

do {
    if (next_id == max_id) {
        next_id = min_id;
    }
    else {
        next_id = next_id + id_inc;
    }

    if (check_suitable_id(next_id))
        return next_id;

    count--;
} while (count > 0);

return ERROR;
```

Note:

check_suitable_id() is a function that checks whether the resulting identifier is acceptable (e.g., whether its in use, etc.).

For obvious reasons, this algorithm results in predictable sequences. If a global counter is used (such as "next_id" in the example above), a node that learns one protocol identifier can also learn or guess values employed by past and future protocol instances. On the other hand, when the value of increments is known (such as "1" in this case), an attacker can sample two values, and learn the number of identifiers that were generated in-between.

Where identifier reuse would lead to a hard failure, one typical approach to generate unique identifiers (while minimizing the security and privacy implications of predictable identifiers) is to obfuscate the resulting protocol IDs by either:

- o Replace the global counter with multiple counters (initialized to a random value)

- o Randomizing the "increments"

Avoiding global counters essentially means that learning one identifier for a given context (e.g., one TCP ephemeral port for a given {src IP, Dst IP, Dst Port}) is of no use for learning or guessing identifiers for a different context (e.g., TCP ephemeral ports that involve other peers). However, this may imply keeping one additional variable/counter per context, which may be prohibitive in some environments. The choice of `id_inc` has implications on both the security and privacy properties of the resulting identifiers, but also on the corresponding interoperability properties. On one hand, minimizing the increments (as in "`id_inc = 1`" in our case) generally minimizes the identifier reuse frequency, albeit at increased predictability. On the other hand, if the increments are randomized predictability of the resulting identifiers is reduced, and the information leakage produced by global constant increments is mitigated.

8.4.2. Per-context Counter Algorithm

One possible way to achieve similar (or even lower) identifier reuse frequency while still avoiding predictable sequences would be to employ a per-context counter, as opposed to a global counter. Such an algorithm could be described as follows:


```
/* Initialization at system boot time. Could be random */
id_inc= 1;

/* Identifier selection function */
count = max_id - min_id + 1;

if(lookup_counter(CONTEXT) == ERROR){
    create_counter(CONTEXT);
}

next_id= lookup_counter(CONTEXT);

do {
    if (next_id == max_id) {
        next_id = min_id;
    }
    else {
        next_id = next_id + id_inc;
    }

    if (check_suitable_id(next_id)){
        store_counter(CONTEXT, next_id);
        return next_id;
    }

    count--;

} while (count > 0);

store_counter(CONTEXT, next_id);
return ERROR;
```

NOTE:

lookup_counter() returns the current counter for a given context, or an error condition if such a counter does not exist.

create_counter() creates a counter for a given context, and initializes such counter to a random value.

store_counter() saves (updates) the current counter for a given context.

check_suitable_id() is a function that checks whether the resulting identifier is acceptable (e.g., whether its in use, etc.).

Essentially, whenever a new identifier is to be selected, the algorithm checks whether there is a counter for the

corresponding context. If there is, such counter is incremented to obtain the new identifier, and the new identifier updates the corresponding counter. If there is no counter for such context, a new counter is created and initialized to a random value, and used as the new identifier.

This algorithm produces a per-context counter, which results in one linear function for each context. Since the origin of each "line" is a random value, the resulting values are unknown to an off-path attacker.

This algorithm has the following drawbacks:

- o If, as a result of resource management, the counter for a given context must be removed, the last identifier value used for that context will be lost. Thus, if subsequently an identifier needs to be generated for such context, that counter will need to be recreated and reinitialized to random value, thus possibly leading to reuse/collision of identifiers.
- o If the identifiers are predictable by the destination system (e.g., the destination host represents the context), a vulnerable host might possibly leak to third parties the identifiers used by other hosts to send traffic to it (i.e., a vulnerable Host B could leak to Host C the identifier values that Host A is using to send packets to Host B). Appendix A of [RFC7739] describes one possible scenario for such leakage in detail.

8.4.3. Simple Hash-Based Algorithm

The goal of this algorithm is to produce monotonically-increasing sequences, with a randomized initial value, for each given context. For example, if the identifiers being generated must be unique for each {src IP, dst IP} set, then each possible combination of {src IP, dst IP} should have a corresponding "next_id" value.

Keeping one value for each possible "context" may in many cases be considered too onerous in terms of memory requirements. As a workaround, the following algorithm employs a calculated technique (as opposed to keeping state in memory) to maintain the random offset for each possible context.

In the following algorithm, the function F() provides (statelessly) a random offset for each given context.

```
/* Initialization at system boot time. Could be random. */
counter = 0;

/* Protocol ID selection function */
id_range = max_id - min_id + 1;
offset = F(CONTEXT, secret_key);
count = id_range;

do {
    next_id = min_id +
              (counter + offset) % id_range;

    counter++;

    if(check_suitable_id(next_id))
        return next_id;

    count--;
} while (count > 0);

return ERROR;
```

The function `F()` provides a "per-CONTEXT" fixed offset within the identifier space. Both the 'offset' and 'counter' variables may take any value within the storage type range since we are restricting the resulting identifier to be in the range `[min_id, max_id]` in a similar way as in the algorithm described in Section 8.1.1. This allows us to simply increment the 'counter' variable and rely on the unsigned integer to wrap around.

The function `F()` should be a cryptographic hash function like SHA-256 [FIPS-SHS]. Note: MD5 [RFC1321] is considered unacceptable for `F()` [RFC6151]. CONTEXT is the concatenation of all the elements that define a given context. For example, if this algorithm is expected to produce identifiers that are monotonically-increasing for each set (Source IP Address, Destination IP Address), the CONTEXT should be the concatenation of these two values.

The secret should be chosen to be as random as possible (see [RFC4086] for recommendations on choosing secrets).

It should be noted that, since this algorithm uses a global counter ("counter") for selecting identifiers, if an attacker could, e.g., force a client to periodically establish a new TCP connection to an attacker-controlled machine (or through an attacker-observable routing path), the attacker could substract consecutive source port

values to obtain the number of outgoing TCP connections established globally by the target host within that time period (up to wrap-around issues and five-tuple collisions, of course).

8.4.4. Double-Hash Algorithm

A trade-off between maintaining a single global 'counter' variable and maintaining $2*N$ 'counter' variables (where N is the width of the result of $F()$) could be achieved as follows. The system would keep an array of `TABLE_LENGTH` integers, which would provide a separation of the increment of the 'counter' variable. This improvement could be incorporated into the algorithm from Section 8.4.3 as follows:

```
/* Initialization at system boot time */
for(i = 0; i < TABLE_LENGTH; i++)
    table[i] = random();

id_inc = 1;

/* Protocol ID selection function */
id_range = max_id - min_id + 1;
offset = F(CONTEXT, secret_key1);
index = G(CONTEXT, secret_key2);
count = id_range;

do {
    next_id = min_id + (offset + table[index]) % id_range;
    table[index] = table[index] + id_inc;

    if(check_suitable_id(next_id))
        return next_id;

    count--;
} while (count > 0);

return ERROR;
```

'table[]' could be initialized with random values, as indicated by the initialization code in pseudo-code above. The function $G()$ should be a cryptographic hash function. It should use the same `CONTEXT` as $F()$, and a secret key value to compute a value between 0 and $(TABLE_LENGTH-1)$. Alternatively, $G()$ could take an "offset" as input, and perform the exclusive-or (XOR) operation between all the bytes in 'offset'.

The array 'table[]' assures that successive identifiers for a given context will be monotonically-increasing. However, the increments space is separated into TABLE_LENGTH different spaces, and thus identifier reuse frequency will be (probabilistically) lower than that of the algorithm in Section 8.4.3. That is, the generation of identifier for one given context will not necessarily result in increments in the identifiers for other contexts.

It is interesting to note that the size of 'table[]' does not limit the number of different identifier sequences, but rather separates the *increments* into TABLE_LENGTH different spaces. The identifier sequence will result from adding the corresponding entry of 'table[]' to the variable 'offset', which selects the actual identifier sequence (as in the algorithm from Section 8.4.3).

An attacker can perform traffic analysis for any "increment space" into which the attacker has "visibility" -- namely, the attacker can force a node to generate identifiers where G(offset) identifies the target "increment space". However, the attacker's ability to perform traffic analysis is very reduced when compared to the predictable linear identifiers (described in Section 8.4.1) and the hash-based identifiers (described in Section 8.4.3). Additionally, an implementation can further limit the attacker's ability to perform traffic analysis by further separating the increment space (that is, using a larger value for TABLE_LENGTH) and/or by randomizing the increments.

8.4.5. Random-Increments Algorithm

This algorithm offers a middle ground between the algorithms that select ephemeral ports randomly (such as those described in Section 8.1.1 and Section 8.1.2), and those that offer obfuscation but no randomization (such as those described in Section 8.4.3 and Section 8.4.4).

```
/* Initialization code at system boot time. */
next_id = random();          /* Initialization value */
id_inc = 500;                /* Determines the trade-off */

/* Identifier selection function */
id_range = max_id - min_id + 1;

count = id_range;

do {
    /* Random increment */
    next_id = next_id + (random() % id_inc) + 1;

    /* Keep the identifier within acceptable range */
    next_id = min_id + (next_id % id_range);

    if(check_suitable_id(next_id))
        return next_id;

    count--;
} while (count > 0);

return ERROR;
```

This algorithm aims at producing a monotonically increasing sequence of identifiers, while avoiding the use of fixed increments, which would lead to trivially predictable sequences. The value "id_inc" allows for direct control of the trade-off between the level of obfuscation and the ID reuse frequency. The smaller the value of "id_inc", the more similar this algorithm is to a predictable, global monotonically-increasing ID generation algorithm. The larger the value of "id_inc", the more similar this algorithm is to the algorithm described in Section 8.1.1 of this document.

When the identifiers wrap, there is the risk of collisions of identifiers (i.e., identifier reuse). Therefore, "id_inc" should be selected according to the following criteria:

- o It should maximize the wrapping time of the identifier space.
- o It should minimize identifier reuse frequency.
- o It should maximize obfuscation.

Clearly, these are competing goals, and the decision of which value of "id_inc" to use is a trade-off. Therefore, the value of "id_inc"

should be configurable so that system administrators can make the trade-off for themselves.

9. Common Vulnerabilities Associated with Identifiers

This section analyzes common vulnerabilities associated with the generation of identifiers for each of the categories identified in Section 7.

9.1. Category #1: Uniqueness (soft failure)

Possible vulnerabilities associated with identifiers of this category are:

- o Use of trivial algorithms (e.g. global counters) that generate predictable identifiers
- o Use of flawed PRNGs.

Since the only interoperability requirement for these identifiers is uniqueness, the obvious approach to generate them is to employ a PRNG. An implementer should consult [RFC4086] regarding randomness requirements for security, and consult relevant documentation when employing a PRNG provided by the underlying system.

Use algorithms other than PRNGs for generating identifiers of this category is discouraged.

9.2. Category #2: Uniqueness (hard failure)

As noted in Section 8.2 this category typically employs the same algorithms as Category #4, since a monotonically-increasing sequence tends to minimize the identifier reuse frequency. Therefore, the vulnerability analysis of Section 9.4 applies to this case.

9.3. Category #3: Uniqueness, constant within context (soft failure)

There are two main vulnerabilities that may be associated with identifiers of this category:

1. Use algorithms or sources that result in predictable identifiers
2. Employing the same identifier across contexts in which constantcy is not required

At times, an implementation or specification may be tempted to employ a source for the identifier which is known to provide unique values. However, while unique, the associated identifiers may have other

properties such as being predictable or leaking information about the node in question. For example, as noted in [RFC7721], embedding link-layer addresses for generating IPv6 IIDs not only results in predictable values, but also leaks information about the manufacturer of the network interface card.

On the other hand, using an identifier across contexts where constancy is not required can be leveraged for correlation of activities. One of the most trivial examples of this is the use of IPv6 IIDs that are constant across networks (such as IIDs that embed the underlying link-layer address).

9.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)

A simple way to generalize algorithms employed for generating identifiers of Category #4 would be as follows:

```
/* Identifier selection function */
count = max_id - min_id + 1;

do {
    linear(CONTEXT) = linear(CONTEXT) + increment();
    next_id = offset(CONTEXT) + linear(CONTEXT);

    if (check_suitable_id(next_id))
        return next_id;

    count--;
} while (count > 0);

return ERROR;
```

Essentially, an identifier (next_id) is generated by adding a linear function (linear()) to an offset value, which is unknown to the attacker, and constant for given context.

The following aspects of the algorithm should be considered:

- o For the most part, it is the offset() function that results in identifiers that are unpredictable by an off-path attacker. While the resulting sequence will be monotonically-increasing, the use of an offset value that is unknown to the attacker makes the resulting values unknown to the attacker.
- o The most straightforward "stateless" implementation of offset would be that in which offset() is the result of a

cryptographically-secure hash-function that takes the values that identify the context and a "secret" (not shown in the figure above) as arguments.

- o Another possible (but stateful) approach would be to simply generate a random offset and store it in memory, and then look-up the corresponding context when a new identifier is to be selected. The algorithm in Section 8.4.2 is essentially an implementation of this type.
- o The linear function is incremented according to `increment()`. In the most trivial case `increment()` could always return the constant "1". But it could also possibly return small integers such the increments are randomized.

Considering the generic algorithm illustrated above we can identify the following possible vulnerabilities:

- o If the offset value spans more than the necessary context, identifiers could be unnecessarily predictable by other parties, since the offset value would be unnecessarily leaked to them. For example, an implementation that means to produce a per-destination counter but replaces `offset()` with a constant number (i.e., employs a global counter), will unnecessarily result in predictable identifiers.
- o The function `linear()` could be seen as representing the number of identifiers that have so far been generated for a given context. If `linear()` spans more than the necessary context, the "increments" could be leaked to other parties, thus disclosing information about the number of identifiers that have so far been generated. For example, an implementation in which `linear()` is implemented as a single global counter will unnecessarily leak information the number of identifiers that have been produced.
- o `increment()` determines how the `linear()` is incremented for each identifier that is selected. In the most trivial case, `increment()` will return the integer "1". However, an implementation may have `increment()` return a "small" integer value such that even if the current value employed by the generator is guessed (see Appendix A of [RFC7739]), the exact next identifier to be selected will be slightly harder to identify.

10. Security and Privacy Requirements for Identifiers

Protocol specifications that specify identifiers should:

1. Clearly specify the interoperability requirements for selecting the aforementioned identifiers.
2. Provide a security and privacy analysis of the aforementioned identifiers.
3. Recommend an algorithm for generating the aforementioned identifiers that mitigates security and privacy issues, such as those discussed in Section 9.

11. IANA Considerations

There are no IANA registries within this document. The RFC-Editor can remove this section before publication of this document as an RFC.

12. Security Considerations

The entire document is about the security and privacy implications of identifiers.

13. Acknowledgements

The authors would like to thank (in alphabetical order) Steven Bellovin, Joseph Lorenzo Hall, Gre Norcie, and Martin Thomson, for providing valuable comments on earlier versions of this document.

The authors would like to thank Diego Armando Maradona for his magic and inspiration.

14. References

14.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<https://www.rfc-editor.org/info/rfc2460>>.

- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <<https://www.rfc-editor.org/info/rfc4862>>.
- [RFC5722] Krishnan, S., "Handling of Overlapping IPv6 Fragments", RFC 5722, DOI 10.17487/RFC5722, December 2009, <<https://www.rfc-editor.org/info/rfc5722>>.
- [RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <<https://www.rfc-editor.org/info/rfc6151>>.
- [RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <<https://www.rfc-editor.org/info/rfc6528>>.
- [RFC7098] Carpenter, B., Jiang, S., and W. Tarreau, "Using the IPv6 Flow Label for Load Balancing in Server Farms", RFC 7098, DOI 10.17487/RFC7098, January 2014, <<https://www.rfc-editor.org/info/rfc7098>>.
- [RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <<https://www.rfc-editor.org/info/rfc7217>>.

14.2. Informative References

- [Bellovin1989] Bellovin, S., "Security Problems in the TCP/IP Protocol Suite", Computer Communications Review, vol. 19, no. 2, pp. 32-48, 1989, <<https://www.cs.columbia.edu/~smb/papers/ipext.pdf>>.

[Bellovin2002]

Bellovin, S., "A Technique for Counting NATted Hosts",
IMW'02 Nov. 6-8, 2002, Marseille, France, 2002.

[CERT2001]

CERT, "CERT Advisory CA-2001-09: Statistical Weaknesses in
TCP/IP Initial Sequence Numbers", 2001,
<<http://www.cert.org/advisories/CA-2001-09.html>>.

[CPNI-TCP]

Gont, F., "Security Assessment of the Transmission Control
Protocol (TCP)", United Kingdom's Centre for the
Protection of National Infrastructure (CPNI) Technical
Report, 2009, <[http://www.gont.com.ar/papers/
tn-03-09-security-assessment-TCP.pdf](http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf)>.

[FIPS-SHS]

FIPS, "Secure Hash Standard (SHS)", Federal Information
Processing Standards Publication 180-4, March 2012,
<[http://csrc.nist.gov/publications/fips/fips180-4/
fips-180-4.pdf](http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf)>.

[Fyodor2004]

Fyodor, "Idle scanning and related IP ID games", 2004,
<<http://www.insecure.org/nmap/idlescan.html>>.

[Gont2011]

Gont, F., "Hacking IPv6 Networks (training course)", Hack
In Paris 2011 Conference Paris, France, June 2011.

[Gont2012]

Gont, F., "Recent Advances in IPv6 Security", BSDCan 2012
Conference Ottawa, Canada. May 11-12, 2012, May 2012.

[I-D.eddy-rfc793bis-04]

Eddy, W., "Transmission Control Protocol Specification",
draft-eddy-rfc793bis-04 (work in progress), August 2014.

[I-D.gont-6man-flowlabel-security]

Gont, F., "Security Assessment of the IPv6 Flow Label",
draft-gont-6man-flowlabel-security-03 (work in progress),
March 2012.

[I-D.ietf-6man-default-iids]

Gont, F., Cooper, A., Thaler, D., and S. LIU,
"Recommendation on Stable IPv6 Interface Identifiers",
draft-ietf-6man-default-iids-16 (work in progress),
September 2016.

- [I-D.ietf-6man-predictable-fragment-id-08]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-ietf-6man-predictable-fragment-id-08 (work in progress), June 2015.
- [Joncheray1995]
Joncheray, L., "A Simple Active Attack Against TCP", Proc. Fifth Usenix UNIX Security Symposium, 1995.
- [Klein2007]
Klein, A., "OpenBSD DNS Cache Poisoning and Multiple O/S Predictable IP ID Vulnerability", 2007,
<http://www.trusteer.com/files/OpenBSD_DNS_Cache_Poisoning_and_Multiple_OS_Predictable_IP_ID_Vulnerability.pdf>.
- [Morris1985]
Morris, R., "A Weakness in the 4.2BSD UNIX TCP/IP Software", CSTR 117, AT&T Bell Laboratories, Murray Hill, NJ, 1985,
<<https://pdos.csail.mit.edu/~rtm/papers/117.pdf>>.
- [RedHat2011]
RedHat, "RedHat Security Advisory RHSA-2011:1465-1: Important: kernel security and bug fix update", 2011,
<<https://rhn.redhat.com/errata/RHSA-2011-1465.html>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992,
<<https://www.rfc-editor.org/info/rfc1321>>.
- [RFC1948] Bellare, S., "Defending Against Sequence Number Attacks", RFC 1948, DOI 10.17487/RFC1948, May 1996,
<<https://www.rfc-editor.org/info/rfc1948>>.
- [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007,
<<https://www.rfc-editor.org/info/rfc4963>>.
- [RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010,
<<https://www.rfc-editor.org/info/rfc5927>>.

- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <<https://www.rfc-editor.org/info/rfc6056>>.
- [RFC7528] Higgs, P. and J. Piesing, "A Uniform Resource Name (URN) Namespace for the Hybrid Broadcast Broadband TV (HbbTV) Association", RFC 7528, DOI 10.17487/RFC7528, April 2015, <<https://www.rfc-editor.org/info/rfc7528>>.
- [RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <<https://www.rfc-editor.org/info/rfc7707>>.
- [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <<https://www.rfc-editor.org/info/rfc7721>>.
- [RFC7739] Gont, F., "Security Implications of Predictable Fragment Identification Values", RFC 7739, DOI 10.17487/RFC7739, February 2016, <<https://www.rfc-editor.org/info/rfc7739>>.
- [Sanfilippo1998a]
Sanfilippo, S., "about the ip header id", Post to Bugtraq mailing-list, Mon Dec 14 1998, <<http://seclists.org/bugtraq/1998/Dec/48>>.
- [Sanfilippo1998b]
Sanfilippo, S., "Idle scan", Post to Bugtraq mailing-list, 1998, <<http://www.kyuzz.org/antirez/papers/dumbscan.html>>.
- [Sanfilippo1999]
Sanfilippo, S., "more ip id", Post to Bugtraq mailing-list, 1999, <<http://www.kyuzz.org/antirez/papers/moreipid.html>>.
- [Shimomura1995]
Shimomura, T., "Technical details of the attack described by Markoff in NYT", Message posted in USENET's comp.security.misc newsgroup Message-ID: <3g5gkl\$5jl@ariel.sdsc.edu>, 1995, <<http://www.gont.com.ar/docs/post-shimomura-usenet.txt>>.

[Silbersack2005]

Silbersack, M., "Improving TCP/IP security through randomization without sacrificing interoperability", EuroBSDCon 2005 Conference, 2005, <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.4542&rep=rep1&type=pdf>>.

[SUSE2011]

SUSE, "SUSE Security Announcement: Linux kernel security update (SUSE-SA:2011:046)", 2011, <<http://lists.opensuse.org/opensuse-security-announce/2011-12/msg00011.html>>.

[Ubuntu2011]

Ubuntu, "Ubuntu: USN-1253-1: Linux kernel vulnerabilities", 2011, <<http://www.ubuntu.com/usn/usn-1253-1/>>.

[USCERT2001]

US-CERT, "US-CERT Vulnerability Note VU#498440: Multiple TCP/IP implementations may use statistically predictable initial sequence numbers", 2001, <<http://www.kb.cert.org/vuls/id/498440>>.

[Zalewski2001]

Zalewski, M., "Strange Attractors and TCP/IP Sequence Number Analysis", 2001, <<http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>>.

[Zalewski2002]

Zalewski, M., "Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later", 2001, <<http://lcamtuf.coredump.cx/newtcp/>>.

[Zalewski2003]

Zalewski, M., "A new TCP/IP blind data injection technique?", 2003, <<http://lcamtuf.coredump.cx/ipfrag.txt>>.

Authors' Addresses

Fernando Gont
SI6 Networks / UTN-FRH
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: fgont@si6networks.com
URI: <http://www.si6networks.com>

Ivan Arce
Quarkslab

Email: iarce@quarkslab.com
URI: <https://www.quarkslab.com>

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: January 9, 2017

R. Salz
Akamai Technologies
July 08, 2016

No MTI Crypto without Public Review
draft-rsalz-drbg-speck-wap-wep-01

Abstract

Cryptography is becoming more important to the IETF and its protocols, and more IETF protocols are using, or looking at, cryptography to increase privacy on the Internet [RFC7258].

This document specifies a proposed best practice for any mechanism (or data format) that uses cryptography; namely, that RFCs cannot specify an algorithm as mandatory-to-implement (MTI) unless that algorithm has had reasonable public review. This document also "sketches out" a rough definition around what such a review would look like.

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 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology	2
2. Introduction	2
3. Why is Cryptography Hard?	3
4. Things to avoid	4
5. Why limit to MTI?	4
6. How to Do it Right	5
6.1. Public Review	6
7. Acknowledgements	6
8. References	6
8.1. Normative References	6
8.2. Informative References	6
Author's Address	7

1. Terminology

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 [RFC2119].

The term mandatory to implement (MTI) is used in this document to describe a cryptographic algorithm that is listed as a MUST in an RFC.

The term "snake oil" is used as a pejorative for something which appears to do its job acceptably, but actually does not; see https://en.wikipedia.org/wiki/Snake_oil_%28cryptography%29 . It is a goal of the IETF that we never be misled into being, or mistakenly taken as, snake oil salesman.

2. Introduction

Cryptography is becoming more important to the IETF and its protocols, and more IETF protocols are using, or looking at, cryptography to increase privacy on the Internet [RFC7258].

This document specifies a proposed best practice for any protocol (or data format) that uses cryptography. Namely, that such RFCs cannot specify an algorithm as mandatory-to-implement (MTI) unless that algorithm has had reasonable public review. This document also

"sketches out" a rough definition around what such a review would look like.

3. Why is Cryptography Hard?

Cryptography is hard because it is not like traditional IETF protocol deployments. In this classic situation, if one party implements a protocol incorrectly, it usually becomes obvious as interoperability suffers or completely fails. But with cryptography, one party can have implementation defects, or known exploitable weaknesses, that expose the entire communication stream to an attacker. Open source and code reviews are not a panacea here, but using only widely-accepted cryptographic mechanisms (e.g., avoiding facilities like https://en.wikipedia.org/wiki/Dual_EC_DRBG) will reduce the attack surface.

Cryptography is hard because subtle design characteristics can have disastrous consequences. For example, the US Digital Signature Algorithm requires the random nonce to be protected and never re-used. If those requirements are not met, the private key can be leaked.

Cryptography is hard because adversaries design new attacks and refine existing ones. Attacks get better over time; they never get worse. For example, it is now de rigueur to protect against CPU timing attacks, even when the device is only viewable over a network. A recent paper [acoustic] (XXX reference) can identify a private key if your smartphone is just laid next to an innocuous charging device. We understand power differential attacks, timing attacks, and perhaps cache line attacks; we now have to think about RFI emissions from our phone.

Cryptography is hard because the order of operations can matter. It is not intuitively obvious to most developers, which should come first among signing, compression and encryption. This issues was first raised in Spring of 2001 [davis] but was only addressed in TLS by [RFC7366] more than a dozen years later.

Getting the cryptography right is important because the Internet, and therefore the work of the IETF, has become a tempting target for all types of attackers, from individual "script kiddies," through criminal commercial botnet and phishing ventures, up to national-scale adversaries operating on behalf of their nation-state.

4. Things to avoid

"Sunlight is said to be the best of disinfectants; electric light the most efficient policeman." - Louis Brandeis, *Other People's Money and How Bankers Use it*, first published as a set of articles in *Harper's Weekly* in 1914.

Cryptography that is developed in private, such as among an industry consortium is a bad idea. Notable examples of this include:

- o A5/1 and A5/2 for GSM-based mobile phones.
- o WEP and WPA for WiFi access.
- o SSLv2, while published, was developed by a private group at an Internet startup. It had security flaws that had global effects decades later, see <https://drownattack.com/>.

It is hard to get good public review of patented cryptography, unless there is a strongly compelling need. For example, decades ago RSA was the only practical public-key mechanism available and it was therefore studied pretty extensively.

Part of the concern about patented cryptography is that the patent-holder has every incentive to provide that their system is good, while the rest of the world generally has little interest in proving that their commercial venture is bad. Examples of this include:

- o Algebraic Eraser, prior to its presentation at IETF-xx, received little public interest.
- o There is not a great deal of study about NTRU.

Both of these items are "lattice cryptography" and that might also be a reason for lack of review; the field might not have much interest yet.

- o XXX STILL MORE NEEDED

5. Why limit to MTI?

There is an argument that any new RFC not classified as "historical" should not specify or recommend insufficiently-reviewed cryptography, whether it MTI or not. This document limits itself to MTI for a couple of reasons.

- o Informational RFCs often document how to interoperate with other systems, and this is useful. As examples of this, see the Internet-Drafts on scrypt and [RFC7693].
- o Putting insufficiently-reviewed algorithms into an RFC can be one way to spur interest in getting more reviews. This MUST NOT be the primary motivation for inclusion, but it can be a useful side-effect, and might lead to future "promotion" to MTI. Note that waiting through draft and last-call state, then claiming "nobody broke it" MUST NOT be used as the rationale; this is using the IETF to host a "proof by contest."
- o Drawing a strict boundary just around MTI is a tractable problem. Drawing a similar boundary around all potential IETF uses of cryptography is bound to have mistakes and errors, any one of which can have the potential to make the IETF look bad, if not incompetent.
- o Requiring MTI to have public review also pressures everyone to conform and raise the bar. Imagine a hypothetical national security body that has a new cryptographic algorithm, Military Top-secret Encryption, or MITE. If MITE is not MTI, then that government might be hard-pressed to get it accepted into off-the-shell offerings. If it is MTI without sufficient review, then they have good reason to keep flaws in existing cryptography private. To avoid both situations, the that government should work to get MITE as an MTI, and would now have the burden to make sure it receives sufficient analysis.

6. How to Do it Right

Cryptographic agility, [RFC7696], is probably a MUST. While it has its detractors, there are no known (to the author) practical considerations to evolving a deployed based to stronger crypto, while still maintaining interoperability with existing entities. This requires being able to make informed choices about when to use old weak crypto, and when to use the "latest and greatest," and while not much software, and essentially no end-users, are capable of making that choice, it seems sadly the best we can do.

NIST is an important reference for crypto algorithms. Yes, they have made mistakes (DUAL_EC_DRBG), but so has the IETF (opaque-prf) in the same area. But they have run respected international contests and their output receives heavy scrutiny.

The second consideration is to avoid temptation and premature optimization. Do not adopt an algorithm just because it seems "small and fast" or comes from "someone I respect."

6.1. Public Review

What constitutes sufficient public review? It is hard to say. This section attempts to provide some guidelines.

An open competition, such as those that led to AES (XXX ref) and SHA-3 (XXX ref) seem to be good, even when they come from sources that are under widespread suspicion, like the US Government. These efforts, like the Password Hashing Competition <https://password-hashing.net/>, had wide international participation and analysis by many noted experts.

Papers presented in the various Crypto conferences (XXX need list) are good. Same for various Usenix workshops.

Proof by contest - "Nobody's Claimed my \$200 reward" - are generally useless, for a number of reasons. They tend to be promoted by amateur cryptographers as a way to get attention, and if someone actually looks at them they are always cracked. Numerical analysis is a better approach, albeit much harder work. Contests designed to show the amount of "brute-force" work needed, such as the old RSA factoring challenges, can be useful. But they do not show, for example, if the cryptography under test is fundamentally flawed or not.

Public review is also a natural fit for the IETF, which takes "rough consensus and running code" as an axiom. Theory reduced to practice is much easier, and much less of a limited academic exercise, to review.

7. Acknowledgements

Thanks to Stephen Farrell for instigating this.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

8.2. Informative References

- [acoustic] Technion and Tel Aviv University, Weizmann Institute of Science, and Tel Aviv University, "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis", December 2013, <<http://www.tau.ac.il/~tromer/papers/acoustic-20131218.pdf>>.
- [davis] "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML.", Usenix Proc. Usenix Tech. Conf., June 2001, <http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.PDF>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<http://www.rfc-editor.org/info/rfc7258>>.
- [RFC7366] Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, DOI 10.17487/RFC7366, September 2014, <<http://www.rfc-editor.org/info/rfc7366>>.
- [RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <<http://www.rfc-editor.org/info/rfc7693>>.
- [RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <<http://www.rfc-editor.org/info/rfc7696>>.

Author's Address

Rich Salz
Akamai Technologies

Email: rsalz@akamai.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 2, 2019

J. Vcelak
CZ.NIC
S. Goldberg
Boston University
D. Papadopoulos
HKUST
S. Huque
Salesforce
D. Lawrence
Dyn
December 29, 2018

NSEC5, DNSSEC Authenticated Denial of Existence
draft-vcclak-nsec5-08

Abstract

The Domain Name System Security Extensions (DNSSEC) introduced two resource records (RR) for authenticated denial of existence: the NSEC RR and the NSEC3 RR. This document introduces NSEC5 as an alternative mechanism for DNSSEC authenticated denial of existence. NSEC5 uses verifiable random functions (VRFs) to prevent offline enumeration of zone contents. NSEC5 also protects the integrity of the zone contents even if an adversary compromises one of the authoritative servers for the zone. Integrity is preserved because NSEC5 does not require private zone-signing keys to be present on all authoritative servers for the zone, in contrast to DNSSEC online signing schemes like NSEC3 White Lies.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <<https://github.com/fcelda/nsec5-draft>>. The most recent version of the document, open issues, etc should all be available there. The authors gratefully accept pull requests.

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 <https://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 July 2, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Rationale	3
1.2. Requirements	6
1.3. Terminology	6
2. Backward Compatibility	6
3. How NSEC5 Works	7
4. NSEC5 Algorithms	8
5. The NSEC5KEY Resource Record	9
5.1. NSEC5KEY RDATA Wire Format	9
5.2. NSEC5KEY RDATA Presentation Format	9
6. The NSEC5 Resource Record	9
6.1. NSEC5 RDATA Wire Format	10
6.2. NSEC5 Flags Field	10
6.3. NSEC5 RDATA Presentation Format	11
7. The NSEC5PROOF Resource Record	11
7.1. NSEC5PROOF RDATA Wire Format	11
7.2. NSEC5PROOF RDATA Presentation Format	12
8. Types of Authenticated Denial of Existence with NSEC5	12
8.1. Name Error Responses	12
8.2. No Data Responses	13
8.2.1. No Data Response, Opt-Out Not In Effect	13

8.2.2. No Data Response, Opt-Out In Effect	14
8.3. Wildcard Responses	14
8.4. Wildcard No Data Responses	14
9. Authoritative Server Considerations	15
9.1. Zone Signing	15
9.1.1. Precomputing Closest Provable Encloser Proofs	16
9.2. Zone Serving	17
9.3. NSEC5KEY Rollover Mechanism	18
9.4. Secondary Servers	18
9.5. Zones Using Unknown NSEC5 Algorithms	18
9.6. Dynamic Updates	18
10. Resolver Considerations	19
11. Validator Considerations	19
11.1. Validating Responses	19
11.2. Validating Referrals to Unsigned Subzones	20
11.3. Responses With Unknown NSEC5 Algorithms	20
12. Special Considerations	20
12.1. Transition Mechanism	20
12.2. Private NSEC5 keys	20
12.3. Domain Name Length Restrictions	21
13. Implementation Status	21
14. Performance Considerations	21
15. Security Considerations	21
15.1. Zone Enumeration Attacks	21
15.2. Compromise of the Private NSEC5 Key	22
15.3. Key Length Considerations	22
15.4. NSEC5 Hash Collisions	22
16. IANA Considerations	23
17. Contributors	23
18. References	24
18.1. Normative References	24
18.2. Informative References	25
Appendix A. Examples	27
A.1. Name Error Example	27
A.2. No Data Example	29
A.3. Delegation to an Unsigned Zone in an Opt-Out span Example	30
A.4. Wildcard Example	31
A.5. Wildcard No Data Example	32
Appendix B. Change Log	33
Authors' Addresses	34

1. Introduction

1.1. Rationale

NSEC5 provides an alternative mechanism for authenticated denial of existence for the DNS Security Extensions (DNSSEC). NSEC5 has two key security properties. First, NSEC5 protects the integrity of the

zone contents even if an adversary compromises one of the authoritative servers for the zone. Second, NSEC5 prevents offline zone enumeration, where an adversary makes a small number of online DNS queries and then processes them offline in order to learn all of the names in a zone. Zone enumeration can be used to identify routers, servers or other "things" that could then be targeted in more complex attacks. An enumerated zone can also be a source of probable email addresses for spam, or as a "key for multiple WHOIS queries to reveal registrant data that many registries may have legal obligations to protect" [RFC5155].

All other DNSSEC mechanisms for authenticated denial of existence either fail to preserve integrity against a compromised server, or fail to prevent offline zone enumeration.

When offline signing with NSEC is used [RFC4034], an NSEC chain of all existing domain names in the zone is constructed and signed offline. The chain is made of resource records (RRs), where each RR represents two consecutive domain names in canonical order present in the zone. The authoritative server proves the non-existence of a name by presenting a signed NSEC RR which covers the name. Because the authoritative server does not need to know the private zone-signing key, the integrity of the zone is protected even if the server is compromised. However, the NSEC chain allows for easy zone enumeration: N queries to the server suffice to learn all N names in the zone (see e.g., [nmap-nsec-enum], [nsec3map], and [ldns-walk]).

When offline signing with NSEC3 is used [RFC5155], the original names in the NSEC chain are replaced by their cryptographic hashes. Offline signing ensures that NSEC3 preserves integrity even if an authoritative server is compromised. However, offline zone enumeration is still possible with NSEC3 (see e.g., [nsec3walker], [nsec3gpu]), and is part of standard network reconnaissance tools (e.g., [nmap-nsec3-enum], [nsec3map]).

When online signing is used, the authoritative server holds the private zone-signing key and uses this key to synthesize NSEC or NSEC3 responses on the fly (e.g. NSEC3 White Lies (NSEC3-WL) or Minimally-Covering NSEC, both described in [RFC7129]). Because the synthesized response only contains information about the queried name (but not about any other name in the zone), offline zone enumeration is not possible. However, because the authoritative server holds the private zone-signing key, integrity is lost if the authoritative server is compromised.

Scheme	Integrity vs network attacks?	Integrity vs compromised auth. server?	Prevents offline zone enumeration?	Online crypto?
Unsigned	NO	NO	YES	NO
NSEC	YES	YES	NO	NO
NSEC3	YES	YES	NO	NO
NSEC3-WL	YES	NO	YES	YES
NSEC5	YES	YES	YES	YES

NSEC5 prevents offline zone enumeration and also protects integrity even if a zone's authoritative server is compromised. To do this, NSEC5 replaces the unkeyed cryptographic hash function used in NSEC3 with a verifiable random function (VRF) [I-D.irtf-cfrg-vrf] [MRV99]. A VRF is the public-key version of a keyed cryptographic hash. Only the holder of the private VRF key can compute the hash, but anyone with public VRF key can verify the correctness of the hash.

The public VRF key is distributed in an NSEC5KEY RR, similar to a DNSKEY RR, and is used to verify NSEC5 hash values. The private VRF key is present on all authoritative servers for the zone, and is used to compute hash values. For every query that elicits a negative response, the authoritative server hashes the query on the fly using the private VRF key, and also returns the corresponding precomputed NSEC5 record(s). In contrast to the online signing approach [RFC7129], the private key that is present on all authoritative servers for NSEC5 cannot be used to modify the zone contents.

Like online signing approaches, NSEC5 requires the authoritative server to perform online public key cryptographic operations for every query eliciting a denying response. This is necessary; [nsec5] proved that online cryptography is required to prevent offline zone enumeration while still protecting the integrity of zone contents against network attacks.

NSEC5 is not intended to replace NSEC or NSEC3. It is an alternative mechanism for authenticated denial of existence. This document specifies NSEC5 based on the VRFs in [I-D.irtf-cfrg-vrf] over the FIPS 186-3 P-256 elliptic curve and over the the Ed25519 elliptic curve. A formal cryptographic proof of security for NSEC5 is in [nsec5ecc].

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], and [RFC4035]; subsequent RFCs that update them in [RFC2136], [RFC2181], [RFC2308], [RFC5155], and [RFC7129]; and DNS terms in [RFC7719].

The reader should also be familiar with verifiable random functions (VRFs) as defined in [I-D.irtf-cfrg-vrf].

The following terminology is used through this document:

Base32hex: The "Base 32 Encoding with Extended Hex Alphabet" as specified in [RFC4648]. The padding characters ("=") are not used in the NSEC5 specification.

Base64: The "Base 64 Encoding" as specified in [RFC4648].

QNAME: The domain name being queried (query name).

Private NSEC5 key: The private key for the verifiable random function (VRF).

Public NSEC5 key: The public key for the VRF.

NSEC5 proof: A VRF proof. The holder of the private NSEC5 key (e.g., authoritative server) can compute the NSEC5 proof for an input domain name. Anyone who knows the public VRF key can verify that the NSEC5 proof corresponds to the input domain name.

NSEC5 hash: A cryptographic digest of an NSEC5 proof. If the NSEC5 proof is known, anyone can compute its corresponding NSEC5 hash.

NSEC5 algorithm: A triple of VRF algorithms that compute an NSEC5 proof (VRF_prove), verify an NSEC5 proof (VRF_verify), and process an NSEC5 proof to obtain its NSEC5 hash (VRF_proof2hash).

2. Backward Compatibility

The specification describes a protocol change that is not backward compatible with [RFC4035] and [RFC5155]. An NSEC5-unaware resolver will fail to validate responses introduced by this document.

To prevent NSEC5-unaware resolvers from attempting to validate the responses, new DNSSEC algorithm identifiers are introduced in Section 16 which alias existing algorithm numbers. The zones signed according to this specification MUST use only these algorithm identifiers, thus NSEC5-unaware resolvers will treat the zone as insecure.

3. How NSEC5 Works

With NSEC5, the original domain name is hashed using a VRF [I-D.irtf-cfrg-vrf] using the following steps:

1. The domain name is processed using a VRF keyed with the private NSEC5 key to obtain the NSEC5 proof. Anyone who knows the public NSEC5 key, normally acquired via an NSEC5KEY RR, can verify that a given NSEC5 proof corresponds to a given domain name.
2. The NSEC5 proof is then processed using a publicly-computable VRF proof2hash function to obtain the NSEC5 hash. The NSEC5 hash can be computed by anyone who knows the input NSEC5 proof.

The NSEC5 hash determines the position of a domain name in an NSEC5 chain.

To sign a zone, the private NSEC5 key is used to compute the NSEC5 hashes for each name in the zone. These NSEC5 hashes are sorted in canonical order [RFC4034], and each consecutive pair forms an NSEC5 RR. Each NSEC5 RR is signed offline using the private zone-signing key. The resulting signed chain of NSEC5 RRs is provided to all authoritative servers for the zone, along with the private NSEC5 key.

To prove non-existence of a particular domain name in response to a query, the server uses the private NSEC5 key to compute the NSEC5 proof and NSEC5 hash corresponding to the queried name. The server then identifies the NSEC5 RR that covers the NSEC5 hash, and responds with this NSEC5 RR and its corresponding RRSIG signature RRset, as well as a synthesized NSEC5PROOF RR that contains the NSEC5 proof corresponding to the queried name.

To validate the response, the client verifies the following items:

- o The client uses the public NSEC5 key, normally acquired from the NSEC5KEY RR, to verify that the NSEC5 proof in the NSEC5PROOF RR corresponds to the queried name.
- o The client uses the VRF proof2hash function to compute the NSEC5 hash from the NSEC5 proof in the NSEC5PROOF RR. The client verifies that the NSEC5 hash is covered by the NSEC5 RR.

- o The client verifies that the NSEC5 RR is validly signed by the RRSIG RRset.

4. NSEC5 Algorithms

The algorithms used for NSEC5 authenticated denial are independent of the algorithms used for DNSSEC signing. An NSEC5 algorithm defines how the NSEC5 proof and the NSEC5 hash are computed and validated.

The NSEC5 proof corresponding to a name is computed using `ECVRF_prove()`, as specified in [I-D.irtf-cfrg-vrf]. The input to `ECVRF_prove()` is a public NSEC5 key followed by a private NSEC5 key followed by an RR owner name in [RFC4034] canonical wire format. The output NSEC5 proof is an octet string.

An NSEC5 hash corresponding to a name is computed from its NSEC5 proof using `ECVRF_proof2hash()`, as specified in [I-D.irtf-cfrg-vrf]. The input to `VRF_proof2hash()` is an NSEC5 proof as an octet string. The output NSEC5 hash is either an octet string, or `INVALID`.

An NSEC5 proof for a name is verified using `ECVRF_verify()`, as specified in [I-D.irtf-cfrg-vrf]. The input is the NSEC5 public key, followed by an NSEC5 proof as an octet string, followed by an RR owner name in [RFC4034] canonical wire format. The output is either `VALID` or `INVALID`.

This document defines the EC-P256-SHA256 NSEC5 algorithm as follows:

- o The VRF is the ECVRF algorithm using the ECVRF-P256-SHA256 ciphersuite specified in [I-D.irtf-cfrg-vrf].
- o The public key format to be used in the NSEC5KEY RR is defined in Section 4 of [RFC6605] and thus is the same as the format used to store ECDSA public keys in DNSKEY RRs.
[NOTE: This specification does not compress the elliptic curve point used for the public key, but we do compress curve points in every other place we use them. The NSEC5KEY record can be shrunk by 31 additional octets by encoding the public key with point compression.]

This document defines the EC-ED25519-SHA512 NSEC5 algorithm as follows:

- o The VRF is the EC-VRF algorithm using the ECVRF-ED25519-SHA512 ciphersuite specified in [I-D.irtf-cfrg-vrf].

- o The public key format to be used in the NSEC5KEY RR is defined in Section 3 of [RFC8080] and thus is the same as the format used to store Ed25519 public keys in DNSKEY RRs.

[NOTE: Could alternatively have the EC-ED25519-SHA512 NSEC5 ciphersuite use the EC-VRF-ED25519-SHA512-ELLIGATOR2 ciphersuite specified in [I-D.irtf-cfrg-vrf].]

5. The NSEC5KEY Resource Record

The NSEC5KEY RR stores a public NSEC5 key. The key allows clients to validate an NSEC5 proof sent by a server.

5.1. NSEC5KEY RDATA Wire Format

The RDATA for the NSEC5KEY RR is as shown below:

```

          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Algorithm   |                               Public Key           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Algorithm is a single octet identifying the NSEC5 algorithm.

Public Key is a variable-sized field holding public key material for NSEC5 proof verification.

5.2. NSEC5KEY RDATA Presentation Format

The presentation format of the NSEC5KEY RDATA is as follows:

The Algorithm field is represented as an unsigned decimal integer.

The Public Key field is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

6. The NSEC5 Resource Record

The NSEC5 RR provides authenticated denial of existence for an RRset or domain name. One NSEC5 RR represents one piece of an NSEC5 chain, proving existence of the owner name and non-existence of other domain names in the part of the hashed domain space that is covered until the next owner name hashed in the RDATA.

6.1. NSEC5 RDATA Wire Format

The RDATA for the NSEC5 RR is as shown below:

```

      1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Key Tag           |           Flags           | Next Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Next Hashed Owner Name           |               /
+-----+-----+-----+-----+-----+-----+-----+-----+
/                                     Type Bit Maps                                     /
+-----+-----+-----+-----+-----+-----+-----+-----+

```

The Key Tag field contains the key tag value of the NSEC5KEY RR that validates the NSEC5 RR, in network byte order. The value is computed from the NSEC5KEY RDATA using the same algorithm used to compute key tag values for DNSKEY RRs. This algorithm is defined in [RFC4034].

The Flags field is a single octet. The meaning of individual bits of the field is defined in Section 6.2.

The Next Length field is an unsigned single octet specifying the length of the Next Hashed Owner Name field in octets.

The Next Hashed Owner Name field is a sequence of binary octets. It contains an NSEC5 hash of the next domain name in the NSEC5 chain.

Type Bit Maps is a variable-sized field encoding RR types present at the original owner name matching the NSEC5 RR. The format of the field is equivalent to the format used in the NSEC3 RR, described in [RFC5155].

6.2. NSEC5 Flags Field

The following one-bit NSEC5 flags are defined:

```

    0 1 2 3 4 5 6 7
+-----+-----+-----+-----+
|           |W|O|
+-----+-----+-----+-----+

```

O - Opt-Out flag

W - Wildcard flag

All the other flags are reserved for future use and MUST be zero.

The Opt-Out flag has the same semantics as in NSEC3. The definition and considerations in [RFC5155] are valid, except that NSEC3 is replaced by NSEC5.

The Wildcard flag indicates that a wildcard synthesis is possible at the original domain name level (i.e., there is a wildcard node immediately descending from the immediate ancestor of the original domain name). The purpose of the Wildcard flag is to reduce the maximum number of RRs required for an authenticated denial of existence proof from (at most) three to (at most) two, as originally described in [I-D.gieben-nsec4] Section 7.2.1.

6.3. NSEC5 RDATA Presentation Format

The presentation format of the NSEC5 RDATA is as follows:

The Key Tag field is represented as an unsigned decimal integer.

The Flags field is represented as an unsigned decimal integer.

The Next Length field is not represented.

The Next Hashed Owner Name field is represented as a sequence of case-insensitive Base32hex digits without any whitespace and without padding.

The Type Bit Maps representation is equivalent to the representation used in NSEC3 RR, described in [RFC5155].

7. The NSEC5PROOF Resource Record

The NSEC5PROOF record is not to be included in the zone file. The NSEC5PROOF record contains the NSEC5 proof, proving the position of the owner name in an NSEC5 chain.

7.1. NSEC5PROOF RDATA Wire Format

The RDATA for the NSEC5PROOF RR is shown below:

```

      1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               |                               |
|                               |                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
      Key Tag                               Owner Name Hash      /
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Key Tag field contains the key tag value of the NSEC5KEY RR that validates the NSEC5PROOF RR, in network byte order.

Owner Name Hash is a variable-sized sequence of binary octets encoding the NSEC5 proof of the owner name of the RR.

7.2. NSEC5PROOF RDATA Presentation Format

The presentation format of the NSEC5PROOF RDATA is as follows:

The Key Tag field is represented as an unsigned decimal integer.

The Owner Name Hash is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

8. Types of Authenticated Denial of Existence with NSEC5

This section summarizes all possible types of authenticated denial of existence. For each type the following lists are included:

1. Facts to prove: the minimum amount of information that an authoritative server must provide to a client to assure the client that the response content is valid.
2. Authoritative server proofs: the names for which the NSEC5PROOF RRs are synthesized and added into the response along with the NSEC5 RRs matching or covering each such name. These records together prove the listed facts.
3. Validator checks: the individual checks that a validating server is required to perform on a response. The response content is considered valid only if all of the checks pass.

If NSEC5 is said to match a domain name, the owner name of the NSEC5 RR has to be equivalent to an NSEC5 hash of that domain name. If an NSEC5 RR is said to cover a domain name, the NSEC5 hash of the domain name must sort in canonical order between that NSEC5 RR's Owner Name and Next Hashed Owner Name.

8.1. Name Error Responses

Facts to prove:

Non-existence of the domain name that explicitly matches the QNAME.

Non-existence of the wildcard that matches the QNAME.

Authoritative server proofs:

NSEC5PROOF for closest encloser and matching NSEC5 RR.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Closest enclosure is in the zone.

The NSEC5 RR matching the closest enclosure has its Wildcard flag cleared.

The NSEC5 RR matching the closest enclosure does not have NS without SOA in the Type Bit Map.

The NSEC5 RR matching the closest enclosure does not have DNAME in the Type Bit Map.

Next closer name is not in the zone.

8.2. No Data Responses

The processing of a No Data response for DS QTYPE differs if the Opt-Out is in effect. For DS QTYPE queries, the validator has two possible checking paths. The correct path can be simply decided by inspecting if the NSEC5 RR in the response matches the QNAME.

Note that the Opt-Out is valid only for DS QTYPE queries.

8.2.1. No Data Response, Opt-Out Not In Effect

Facts to prove:

Existence of an RRset explicitly matching the QNAME.

Non-existence of QTYPE RRset matching the QNAME.

Non-existence of CNAME RRset matching the QNAME.

Authoritative server proofs:

NSEC5PROOF for the QNAME and matching NSEC5 RR.

Validator checks:

QNAME is in the zone.

NSEC5 RR matching the QNAME does not have QTYPE in Type Bit Map.

NSEC5 RR matching the QNAME does not have CNAME in Type Bit Map.

8.2.2. No Data Response, Opt-Out In Effect

Facts to prove:

The delegation is not covered by the NSEC5 chain.

Authoritative server proofs:

NSEC5PROOF for closest provable enclosure and matching NSEC5 RR.

Validator checks:

Closest provable enclosure is in zone.

Closest provable enclosure covers (not matches) the QNAME.

NSEC5 RR matching the closest provable enclosure has Opt-Out flag set.

8.3. Wildcard Responses

Facts to prove:

A signed positive response to the QNAME demonstrating the existence of the wildcard (label count in RRSIG is less than in QNAME), and also providing closest enclosure name.

Non-existence of the domain name matching the QNAME.

Authoritative server proofs:

A signed positive response for the wildcard expansion of the QNAME.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Next closer name is not in the zone.

8.4. Wildcard No Data Responses

Facts to prove:

The existence of the wildcard at the closest enclosure to the QNAME.

Non-existence of both the QTYPE and of the CNAME type that matches QNAME via wildcard expansion.

Authoritative server proofs:

NSEC5PROOF for source of synthesis (i.e., wildcard at closest encloser) and matching NSEC5 RR.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Closest encloser to the QNAME exists.

NSEC5 RR matching the wildcard label prepended to the closest encloser, and which does not have the bits corresponding to the QTYPE and CNAME types set in the type bitmap.

9. Authoritative Server Considerations

9.1. Zone Signing

Zones using NSEC5 MUST satisfy the same properties as described in Section 7.1 of [RFC5155], with NSEC3 replaced by NSEC5. In addition, the following conditions MUST be satisfied as well:

- o If the original owner name has a wildcard label immediately descending from the original owner name, the corresponding NSEC5 RR MUST have the Wildcard flag set in the Flags field. Otherwise, the flag MUST be cleared.
- o The zone apex MUST include an NSEC5KEY RRset containing a NSEC5 public key allowing verification of the current NSEC5 chain.

The following steps describe one possible method to properly add required NSEC5 related records into a zone. This is not the only such existing method.

1. Select an algorithm for NSEC5 and generate the public and private NSEC5 keys.
2. Add an NSEC5KEY RR into the zone apex containing the public NSEC5 key.
3. For each unique original domain name in the zone and each empty non-terminal, add an NSEC5 RR. If Opt-Out is used, owner names of unsigned delegations MAY be excluded.

- A. The owner name of the NSEC5 RR is the NSEC5 hash of the original owner name encoded in Base32hex without padding, prepended as a single label to the zone name.
 - B. Set the Key Tag field to be the key tag corresponding to the public NSEC5 key.
 - C. Clear the Flags field. If Opt-Out is being used, set the Opt-Out flag. If there is a wildcard label directly descending from the original domain name, set the Wildcard flag. Note that the wildcard can be an empty non-terminal (i.e., the wildcard synthesis does not take effect and therefore the flag is not to be set).
 - D. Set the Next Length field to a value determined by the used NSEC5 algorithm. Leave the Next Hashed Owner Name field blank.
 - E. Set the Type Bit Maps field based on the RRsets present at the original owner name.
- 4. Sort the set of NSEC5 RRs into canonical order.
 - 5. For each NSEC5 RR, set the Next Hashed Owner Name field by using the owner name of the next NSEC5 RR in the canonical order. If the updated NSEC5 is the last NSEC5 RR in the chain, the owner name of the first NSEC5 RR in the chain is used instead.

The NSEC5KEY and NSEC5 RRs MUST have the same class as the zone SOA RR. Also the NSEC5 RRs SHOULD have the same TTL value as the SOA minimum TTL field.

Notice that a use of Opt-Out is not indicated in the zone. This does not affect the ability of a server to prove insecure delegations. The Opt-Out MAY be part of the zone-signing tool configuration.

9.1.1. Precomputing Closest Provable Encloser Proofs

Per Section 8, the worst-case scenario when answering a negative query with NSEC5 requires the authoritative server to respond with two NSEC5PROOF RRs and two NSEC5 RRs. One pair of NSEC5PROOF and NSEC5 RRs corresponds to the closest provable encloser, and the other pair corresponds to the next closer name. The NSEC5PROOF corresponding to the next closer name MUST be computed on the fly by the authoritative server when responding to the query. However, the NSEC5PROOF corresponding to the closest provable encloser MAY be precomputed and stored as part of zone signing.

Precomputing NSEC5PROOF RRs can halve the number of online cryptographic computations required when responding to a negative query. NSEC5PROOF RRs MAY be precomputed as part of zone signing as follows: For each NSEC5 RR, compute an NSEC5PROOF RR corresponding to the original owner name of the NSEC5 RR. The content of the precomputed NSEC5PROOF record MUST be the same as if the record was computed on the fly when serving the zone. NSEC5PROOF records are not part of the zone and SHOULD be stored separately from the zone file.

9.2. Zone Serving

This specification modifies DNSSEC-enabled DNS responses generated by authoritative servers. In particular, it replaces use of NSEC or NSEC3 RRs in such responses with NSEC5 RRs and adds NSEC5PROOF RRs.

According to the type of a response, an authoritative server MUST include NSEC5 RRs in the response, as defined in Section 8. For each NSEC5 RR in the response, a corresponding RRSIG RRset and an NSEC5PROOF MUST be added as well. The NSEC5PROOF RR has its owner name set to the domain name required according to the description in Section 8. The class and TTL of the NSEC5PROOF RR MUST be the same as the class and TTL value of the corresponding NSEC5 RR. The RDATA payload of the NSEC5PROOF is set according to the description in Section 7.1.

Notice that the NSEC5PROOF owner name can be a wildcard (e.g., source of synthesis proof in wildcard No Data responses). The name also always matches the domain name required for the proof while the NSEC5 RR may only cover (not match) the name in the proof (e.g., closest enclosure in Name Error responses).

If NSEC5 is used, an answering server MUST use exactly one NSEC5 chain for one signed zone.

NSEC5 MUST NOT be used in parallel with NSEC, NSEC3, or any other authenticated denial of existence mechanism that allows for enumeration of zone contents, as this would defeat a principal security goal of NSEC5.

Similarly to NSEC3, the owner names of NSEC5 RRs are not represented in the NSEC5 chain and therefore NSEC5 records deny their own existence. The desired behavior caused by this paradox is the same as described in Section 7.2.8 of [RFC5155].

9.3. NSEC5KEY Rollover Mechanism

Replacement of the NSEC5 key implies generating a new NSEC5 chain. The NSEC5KEY rollover mechanism is similar to "Pre-Publish Zone Signing Key Rollover" as specified in [RFC6781]. The NSEC5KEY rollover MUST be performed as a sequence of the following steps:

1. A new public NSEC5 key is added into the NSEC5KEY RRset in the zone apex.
2. The old NSEC5 chain is replaced by a new NSEC5 chain constructed using the new key. This replacement MUST happen as a single atomic operation; the server MUST NOT be responding with RRs from both the new and old chain at the same time.
3. The old public key is removed from the NSEC5KEY RRset in the zone apex.

The minimum delay between steps 1 and 2 MUST be the time it takes for the data to propagate to the authoritative servers, plus the TTL value of the old NSEC5KEY RRset.

The minimum delay between steps 2 and 3 MUST be the time it takes for the data to propagate to the authoritative servers, plus the maximum zone TTL value of any of the data in the previous version of the zone.

9.4. Secondary Servers

This document does not define mechanism to distribute private NSEC5 keys. See Section 15.2 for security considerations for private NSEC5 keys.

9.5. Zones Using Unknown NSEC5 Algorithms

Zones that are signed with an unknown NSEC5 algorithm or with an unavailable private NSEC5 key cannot be effectively served. Such zones SHOULD be rejected when loading and servers SHOULD respond with RCODE=2 (Server failure) when handling queries that would fall under such zones.

9.6. Dynamic Updates

A zone signed using NSEC5 MAY accept dynamic updates [RFC2136]. The changes to the zone MUST be performed in a way that ensures that the zone satisfies the properties specified in Section 9.1 at any time. The process described in [RFC5155] Section 7.5 describes how to

handle the issues surrounding the handling of empty non-terminals as well as Opt-Out.

It is RECOMMENDED that the server rejects all updates containing changes to the NSEC5 chain and its related RRSIG RRs, and performs itself any required alternations of the NSEC5 chain induced by the update. Alternatively, the server MUST verify that all the properties are satisfied prior to performing the update atomically.

10. Resolver Considerations

The same considerations as described in Section 9 of [RFC5155] for NSEC3 apply to NSEC5. In addition, as NSEC5 RRs can be validated only with appropriate NSEC5PROOF RRs, the NSEC5PROOF RRs MUST be all together cached and included in responses with NSEC5 RRs.

11. Validator Considerations

11.1. Validating Responses

The validator MUST ignore NSEC5 RRs with Flags field values other than the ones defined in Section 6.2.

The validator MAY treat responses as bogus if the response contains NSEC5 RRs that refer to a different NSEC5KEY.

According to a type of a response, the validator MUST verify all conditions defined in Section 8. Prior to making decision based on the content of NSEC5 RRs in a response, the NSEC5 RRs MUST be validated.

To validate a denial of existence, public NSEC5 keys for the zone are required in addition to DNSSEC public keys. Similarly to DNSKEY RRs, the NSEC5KEY RRs are present at the zone apex.

The NSEC5 RR is validated as follows:

1. Select a correct public NSEC5 key to validate the NSEC5 proof. The Key Tag value of the NSEC5PROOF RR must match with the key tag value computed from the NSEC5KEY RDATA.
2. Validate the NSEC5 proof present in the NSEC5PROOF Owner Name Hash field using the public NSEC5 key. If there are multiple NSEC5KEY RRs matching the key tag, at least one of the keys must validate the NSEC5 proof.
3. Compute the NSEC5 hash value from the NSEC5 proof and check if the response contains NSEC5 RR matching or covering the computed

NSEC5 hash. The TTL values of the NSEC5 and NSEC5PROOF RRs must be the same.

4. Validate the signature on the NSEC5 RR.

If the NSEC5 RR fails to validate, it MUST be ignored. If some of the conditions required for an NSEC5 proof are not satisfied, the response MUST be treated as bogus.

Notice that determining the closest encloser and next closer name in NSEC5 is easier than in NSEC3. NSEC5 and NSEC5PROOF RRs are always present in pairs in responses and the original owner name of the NSEC5 RR matches the owner name of the NSEC5PROOF RR.

11.2. Validating Referrals to Unsigned Subzones

The same considerations as defined in Section 8.9 of [RFC5155] for NSEC3 apply to NSEC5.

11.3. Responses With Unknown NSEC5 Algorithms

A validator MUST ignore NSEC5KEY RRs with unknown NSEC5 algorithms. The practical result of this is that zones signed with unknown algorithms will be considered bogus.

12. Special Considerations

12.1. Transition Mechanism

[TODO: The following information will be covered.]

- o Transition to NSEC5 from NSEC/NSEC3
- o Transition from NSEC5 to NSEC/NSEC3
- o Transition to new NSEC5 algorithms

12.2. Private NSEC5 keys

This document does not define a format to store private NSEC5 keys. Use of a standardized and adopted format is RECOMMENDED.

The private NSEC5 key MAY be shared between multiple zones, however a separate key is RECOMMENDED for each zone.

12.3. Domain Name Length Restrictions

NSEC5 creates additional restrictions on domain name lengths. In particular, zones with names that, when converted into hashed owner names, exceed the 255 octet length limit imposed by [RFC1035] cannot use this specification.

The actual maximum length of a domain name depends on the length of the zone name and the NSEC5 algorithm used.

All NSEC5 algorithms defined in this document use 256-bit NSEC5 hash values. Such a value can be encoded in 52 characters in Base32hex without padding. When constructing the NSEC5 RR owner name, the encoded hash is prepended to the name of the zone as a single label which includes the length field of a single octet. The maximum length of the zone name in wire format using the 256-bit hash is therefore 202 octets (255 - 53).

13. Implementation Status

NSEC5 has been implemented for the Knot DNS authoritative server (version 1.6.4) and the Unbound recursive server (version 1.5.9). The implementations did not introduce additional library dependencies; all cryptographic primitives are already present in OpenSSL v1.0.2j, which is used by both implementations. The implementations support the full spectrum of negative responses, (i.e., NXDOMAIN, NODATA, Wildcard, Wildcard NODATA, and unsigned delegation) using the EC-P256-SHA256 algorithm. The code is deliberately modular, so that the EC-ED25519-SHA256 algorithm could be implemented by using the Ed25519 elliptic curve [RFC8080] as a drop-in replacement for the P256 elliptic curve. The authoritative server implements the optimization from Section 9.1.1 to precompute the NSEC5PROOF RRs matching each NSEC5 record.

14. Performance Considerations

The performance of NSEC5 has been evaluated in [nsec5ecc].

15. Security Considerations

15.1. Zone Enumeration Attacks

NSEC5 is robust to zone enumeration via offline dictionary attacks by any attacker that does not know the private NSEC5 key. Without the private NSEC5 key, that attacker cannot compute the NSEC5 proof that corresponds to a given domain name. The only way it can learn the NSEC5 proof value for a domain name is by querying the authoritative server for that name. Without the NSEC5 proof value, the attacker

cannot learn the NSEC5 hash value. Thus, even an attacker that collects the entire chain of NSEC5 RR for a zone cannot use offline attacks to "reverse" that NSEC5 hash values in these NSEC5 RR and thus learn which names are present in the zone. A formal cryptographic proof of this property is in [nsec5] and [nsec5ecc].

15.2. Compromise of the Private NSEC5 Key

NSEC5 requires authoritative servers to hold the private NSEC5 key, but not the private zone-signing keys or the private key-signing keys for the zone.

The private NSEC5 key cannot be used to modify zone contents, because zone contents are signed using the private zone-signing key. As such, a compromise of the private NSEC5 key does not compromise the integrity of the zone. An adversary that learns the private NSEC5 key can, however, perform offline zone-enumeration attacks. For this reason, the private NSEC5 key need only be as secure as the DNSSEC records whose privacy (against zone enumeration) is being protected by NSEC5. A formal cryptographic proof of this property is in [nsec5] and [nsec5ecc].

To preserve this property of NSEC5, the private NSEC5 key MUST be different from the private zone-signing keys or key-signing keys for the zone.

15.3. Key Length Considerations

The NSEC5 key must be long enough to withstand attacks for as long as the privacy of the zone contents is important. Even if the NSEC5 key is rolled frequently, its length cannot be too short, because zone privacy may be important for a period of time longer than the lifetime of the key. For example, an attacker might collect the entire chain of NSEC5 RR for the zone over one short period, and then, later (even after the NSEC5 key expires) perform an offline dictionary attack that attempts to reverse the NSEC5 hash values present in the NSEC5 RRs. This is in contrast to zone-signing and key-signing keys used in DNSSEC; these keys, which ensure the authenticity and integrity of the zone contents, need to remain secure only during their lifetime.

15.4. NSEC5 Hash Collisions

If the NSEC5 hash of a QNAME collides with the NSEC5 hash of the owner name of an NSEC5 RR, it will be impossible to prove the non-existence of the colliding QNAME. However, the NSEC5 VRFs ensure that obtaining such a collision is as difficult as obtaining a collision in the SHA-256 hash function, requiring approximately 2^{128}

effort. Note that DNSSEC already relies on the assumption that a cryptographic hash function is collision-resistant, since these hash functions are used for generating and validating signatures and DS RRs. See also the discussion on key lengths in [nsec5].

16. IANA Considerations

This document updates the IANA registry "Domain Name System (DNS) Parameters" in subregistry "Resource Record (RR) TYPEs", by defining the following new RR types:

NSEC5KEY value TBD.

NSEC5 value TBD.

NSEC5PROOF value TBD.

This document creates a new IANA registry for NSEC5 algorithms. This registry is named "DNSSEC NSEC5 Algorithms". The initial content of the registry is:

0 is Reserved.

1 is EC-P256-SHA256.

2 is EC-ED25519-SHA256.

3-255 is Available for assignment.

This document updates the IANA registry "DNS Security Algorithm Numbers" by defining following aliases:

TBD is NSEC5-ECDSAP256SHA256 alias for ECDSAP256SHA256 (13).

TBD is NSEC5-ED25519, alias for ED25519 (15).

17. Contributors

This document would not be possible without help of Moni Naor (Weizmann Institute), Sachin Vasant (Cisco Systems), Leonid Reyzin (Boston University), and Asaf Ziv (Weizmann Institute) who contributed to the design of NSEC5. Ondrej Sury (CZ.NIC Labs), and Duane Wessels (Verisign Labs) provided advice on the implementation of NSEC5, and assisted with evaluating its performance.

18. References

18.1. Normative References

- [FIPS-186-3]
National Institute for Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-3, June 2009.
- [I-D.irtf-cfrg-vrf]
Goldberg, S., Reyzin, L., Papadopoulos, D., and J. Vcelak, "Verifiable Random Functions (VRFs)", draft-irtf-cfrg-vrf-03 (work in progress), September 2018.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<https://www.rfc-editor.org/info/rfc2136>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <<https://www.rfc-editor.org/info/rfc2181>>.
- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <<https://www.rfc-editor.org/info/rfc2308>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<https://www.rfc-editor.org/info/rfc4033>>.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <<https://www.rfc-editor.org/info/rfc4034>>.

- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<https://www.rfc-editor.org/info/rfc4035>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, DOI 10.17487/RFC5114, January 2008, <<https://www.rfc-editor.org/info/rfc5114>>.
- [RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <<https://www.rfc-editor.org/info/rfc5155>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, DOI 10.17487/RFC6605, April 2012, <<https://www.rfc-editor.org/info/rfc6605>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RFC8080] Sury, O. and R. Edmonds, "Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC", RFC 8080, DOI 10.17487/RFC8080, February 2017, <<https://www.rfc-editor.org/info/rfc8080>>.

18.2. Informative References

- [I-D.gieben-nsec4]
Gieben, R. and M. Mekking, "DNS Security (DNSSEC) Authenticated Denial of Existence", draft-gieben-nsec4-01 (work in progress), July 2012.
- [ldns-walk]
NLNetLabs, "ldns", 2015, <<http://git.nlnetlabs.nl/ldns/tree/examples/ldns-walk.c>>.

- [MRV99] Michali, S., Rabin, M., and S. Vadhan, "Verifiable Random Functions", in FOCS, 1999.
- [nmap-nsec-enum] Bond, J., "nmap: dns-nsec-enum", 2011, <<https://nmap.org/nsedoc/scripts/dns-nsec-enum.html>>.
- [nmap-nsec3-enum] Nikolic, A. and J. Bond, "nmap: dns-nsec3-enum", 2011, <<https://nmap.org/nsedoc/scripts/dns-nsec3-enum.html>>.
- [nsec3gpu] Wander, M., Schwittmann, L., Boelmann, C., and T. Weis, "GPU-Based NSEC3 Hash Breaking", in IEEE Symp. Network Computing and Applications (NCA), 2014.
- [nsec3map] anonion0, "nsec3map with John the Ripper plugin", 2015, <<https://github.com/anonion0/nsec3map>>.
- [nsec3walker] Bernstein, D., "Nsec3 walker", 2011, <<http://dnscurve.org/nsec3walker.html>>.
- [nsec5] Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., and A. Ziv, "NSEC5: Provably Preventing DNSSEC Zone Enumeration", in NDSS'15, July 2014, <<https://eprint.iacr.org/2014/582.pdf>>.
- [nsec5ecc] Papadopoulos, D., Wessels, D., Huque, S., Vcelak, J., Naor, M., Reyzin, L., and S. Goldberg, "Can NSEC5 be Practical for DNSSEC Deployments?", in ePrint Cryptology Archive 2017/099, February 2017, <<https://eprint.iacr.org/2017/099.pdf>>.
- [RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <<https://www.rfc-editor.org/info/rfc6781>>.
- [RFC7129] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", RFC 7129, DOI 10.17487/RFC7129, February 2014, <<https://www.rfc-editor.org/info/rfc7129>>.
- [RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <<https://www.rfc-editor.org/info/rfc7719>>.

Appendix A. Examples

We use a small DNS zone to illustrate how negative responses are handled with NSEC5. For brevity, the class is not shown (defaults to IN) and the SOA record is shortened, resulting in the following zone file:

```
example.org.      SOA ( ... )
example.org.      NS  a.example.org

a.example.org.    A  192.0.2.1

c.example.org.    A  192.0.2.2
c.example.org.    TXT "c record"

d.example.org.    NS  ns1.d.example.org

ns1.d.example.org. A  192.0.2.4

g.example.org.    A  192.0.2.1
g.example.org.    TXT "g record"

*.a.example.org.  TXT "wildcard record"
```

Notice the delegation to an unsigned zone `d.example.org` served by `ns1.d.example.org`. (Note: if the `d.example.org` zone was signed, then the `example.org` zone have a DS record for `d.example.org`.)

Next we present example responses. All cryptographic values are shortened as indicated by "... " and ADDITIONAL sections have been removed.

A.1. Name Error Example

Consider a query for a type A record for `a.b.c.example.org`.

The server must prove the following facts:

- o Existence of closest encloser `c.example.org`.
- o Non-existence of wildcard at closest encloser `*.c.example.org`.
- o Non-existence of next closer `b.c.example.org`.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NXDOMAIN; id: 5937

;; QUESTION SECTION:
;; a.b.c.example.org.          IN      A

;; AUTHORITY SECTION:
example.org.          3600 IN SOA a.example.org. hostmaster.example.org. (
                        2010111214 21600 3600 604800 86400 )

example.org.          3600 IN RRSIG  SOA 16 2 3600 (
                        20170412024301 20170313024301 5137 example.org. rT231b1rH... )
```

This is an NSEC5PROOF RR for c.example.com. It's RDATA is the NSEC5 proof corresponding to c.example.com. (NSEC5 proofs are randomized values, because NSEC5 proof values are computed uses the EC-VRF from [I-D.irtf-cfrg-vrf].) Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
c.example.org.      86400 IN NSEC5PROOF 48566 Amgn22zUiZ9JVyaT...
```

This is a signed NSEC5 RR "matching" c.example.org, which proves the existence of closest enclosure c.example.org. The NSEC5 RR has its owner name equal to the NSEC5 hash of c.example.org, which is 04K89V. (NSEC5 hash values are deterministic given the public NSEC5 key.) The NSEC5 RR also has its Wildcard flag cleared (see the "0" after the key ID 48566). This proves the non-existence of the wildcard at the closest enclosure *.c.example.com. NSEC5 RRs are precomputed.

```
o4k89v.example.org. 86400 IN NSEC5      48566 0 0049PI A TXT RRSIG
o4k89v.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
                        20170412024301 20170313024301 5137 example.org. zDNTSMQNLz... )
```

This is an NSEC5PROOF RR for b.c.example.org. It's RDATA is the NSEC5 proof corresponding to b.c.example.com. This NSEC5PROOF RR must be computed on the fly.

```
b.c.example.org.      86400 IN NSEC5PROOF 48566 AuvvJqbUcEs8sCpY...
```

This is a signed NSEC5 RR "covering" b.c.example.org, which proves the non-existence of the next closer name b.c.example.org The NSEC5 hash of b.c.example.org, which is AO5OF, sorts in canonical order between the "covering" NSEC5 RR's Owner Name (which is 0049PI) and Next Hashed Owner Name (which is BAPROH).

```
0o49pi.example.org. 86400 IN NSEC5      48566 0 BAPROH (
    NS SOA RRSIG DNSKEY NSEC5KEY )

0o49pi.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
    20170412024301 20170313024301 5137 example.org. 4HT1uj1YlMzO)
```

[TODO: Add discussion of CNAME and DNAME to the example?]

A.2. No Data Example

Consider a query for a type MX record for c.example.org.

The server must prove the following facts:

- o Existence of c.example.org. for any type other than MX or CNAME

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 38781

;; QUESTION SECTION:
;; c.example.org.      IN MX

;; AUTHORITY SECTION:
example.org.      3600 IN SOA      a.example.org. hostmaster.example.org. (
    2010111214 21600 3600 604800 86400 )

example.org.      3600 IN RRSIG      SOA 16 2 3600 20170412024301 20170313024301 5137
example.org. /rt231blrH/p
```

This is an NSEC5PROOF RR for c.example.com. Its RDATA corresponds to the NSEC5 proof for c.example.com. which is a randomized value. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
c.example.org. 86400 IN NSEC5PROOF 48566 Amgn22zUiZ9JVyaT
```

This is a signed NSEC5 RR "matching" c.example.org. with CNAME and MX Type Bits cleared and its TXT Type Bit set. This NSEC5 RR has its owner name equal to the NSEC5 hash of c.example.org. This proves the existence of c.example.org. for a type other than MX and CNAME. NSEC5 RR are precomputed.

```
o4k89v.example.org. 86400 IN NSEC5      48566 0 0049PI A TXT RRSIG

o4k89v.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
    20170412024301 20170313024301 5137 example.org. zDNTSMQNlZ/J)
```

A.3. Delegation to an Unsigned Zone in an Opt-Out span Example

Consider a query for a type A record for foo.d.example.org.

Here, d.example.org is a delegation to an unsigned zone, which lies within an Opt-Out span.

The server must prove the following facts:

- o Non-existence of signature on next closer name d.example.org.
- o Opt-out bit is set in NSEC5 record covering next closer name d.example.org.
- o Existence of closest provable encloser example.org

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 45866
```

```
;; QUESTION SECTION:
;; foo.d.example.org.          IN A
```

```
;; AUTHORITY SECTION:
d.example.org.          3600  IN NS      ns1.d.example.org.
```

This is an NSEC5PROOF RR for d.example.org. Its RDATA is the NSEC5 proof corresponding to d.example.org. This NSEC5PROOF RR is computed on the fly.

```
d.example.org.          86400  IN      NSEC5PROOF      48566 A9FpmeH79q7g6VNW
```

This is a signed NSEC5 RR "covering" d.example.org with its Opt-out bit set (see the "1" after the key ID 48566). The NSEC5 hash of d.example.org (which is BLE8LR) sorts in canonical order between the "covering" NSEC5 RR's Owner Name (BAPROH) and Next Hashed Owner Name (JQBMG4). This proves that no signed RR exists for d.example.org, but that the zone might contain a unsigned RR for a name whose NSEC5 hash sorts in canonical order between BAPROH and JQBMG4.

```
baproh.example.org. 86400 IN NSEC5      48566 1 JQBMG4 A TXT RRSIG
```

```
baproh.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1)
```

This is an NSEC5PROOF RR for example.com. It's RDATA is the NSEC5 proof corresponding to example.com. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
example.org.          86400 IN NSEC5PROOF      48566 AjwsPCJZ8zH/D0Tr
```

This is a signed NSEC5 RR "matching" example.org which proves the existence of a signed RRs for example.org. This NSEC5 RR has its owner name equal to the NSEC5 hash of example.org which is 0049PI. NSEC5 RR are precomputed.

```
0o49pi.example.org. 86400 IN NSEC5      48566 0 BAPROH (
                        NS SOA RRSIG DNSKEY NSEC5KEY)
```

```
0o49pi.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
                        20170412034216 20170313034216 5137 example.org. 4HT1uj1YlMzO)
```

A.4. Wildcard Example

Consider a query for a type TXT record for foo.a.example.org.

The server must prove the following facts:

- o Existence of the TXT record for the wildcard *.a.example.org
- o Non-existence of the next closer name foo.a.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 53731
```

```
;; QUESTION SECTION:
;; foo.a.example.org.      IN TXT
```

This is a signed TXT record for the wildcard at a.example.org (number of labels is set to 3 in the RRSIG record).

```
;; ANSWER SECTION:
foo.a.example.org.      3600 IN TXT      "wildcard record"

foo.a.example.org.      3600 IN RRSIG      TXT 16 3 3600 (
                        20170412024301 20170313024301 5137 example.org. aeaLgZ8sk+98)
```

```
;; AUTHORITY SECTION:
example.org.            3600 IN NS      a.example.org.
```

```
example.org.            3600 IN RRSIG      NS 16 2 3600 (
                        20170412024301 20170313024301 5137 example.org. 8zuN0h2x5WyF)
```

This is an NSEC5PROOF RR for foo.a.example.org. This NSEC5PROOF RR must be computed on-the-fly.

```
foo.a.example.org.      86400 IN NSEC5PROOF      48566 AjqF5FGGVso40Lda
```

This is a signed NSEC5 RR "covering" foo.a.example.org. The NSEC5 hash of foo.a.example.org is FORDMO and sorts in canonical order between the NSEC5 RR's Owner Name (which is BAPROH) and Next Hashed Owner Name (which is JQBMG4). This proves the non-existence of the next closer name foo.a.example.com. NSEC5 RRs are precomputed.

```
baproh.example.org. 86400 IN NSEC5      48566 1 JQBMG4 A TXT RRSIG
baproh.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
20170412024301 20170313024301 5137 example.org. fjtCoRKgdML1
```

A.5. Wildcard No Data Example

Consider a query for a type MX record for foo.a.example.org.

The server must prove the following facts:

- o Existence of wildcard at closest encloser *.a.example.org. for any type other than MX or CNAME.
- o Non-existence of the next closer name foo.a.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 17332
```

```
;; QUESTION SECTION:
```

```
;; foo.a.example.org.          IN          MX
```

```
;; AUTHORITY SECTION:
```

```
example.org.      3600 IN SOA      a.example.org. hostmaster.example.org. (
20101111214 21600 3600 604800 86400 )
```

```
example.org.      3600 IN RRSIG      SOA 16 2 3600 (
20170412024301 20170313024301 5137 example.org. /rT231b1rH/p )
```

This is an NSEC5PROOF RR for *.a.example.com, with RDATA equal to the NSEC5 proof for *.a.example.com. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
*.a.example.org.  86400 IN NSEC5PROOF      48566 Aq38RWWPhbs/vtjh
```

This is a signed NSEC5 RR "matching" *.a.example.org with its CNAME and MX Type Bits cleared and its TXT Type Bit set. This NSEC5 RR has its owner name equal to the NSEC5 hash of *.a.example.org. NSEC5 RRs are precomputed.

```
mpu6c4.example.org. 86400 IN NSEC5      48566 0 04K89V TXT RRSIG
mpu6c4.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
    20170412024301 20170313024301 5137 example.org. m3I75ttcWwVC )
```

This is an NSEC5PROOF RR for foo.a.example.com. This NSEC5PROOF RR must be computed on-the-fly.

```
foo.a.example.org. 86400 IN NSEC5PROOF      48566 AjqF5FGGVso40Lda
```

This is a signed NSEC5 RR "covering" foo.a.example.org. The NSEC5 hash of foo.a.example.org is FORDMO, and sorts in canonical order between this covering NSEC5 RR's Owner Name (which is BAPROH) and Next Hashed Owner Name (which is JQBMG4). This proves the existence of the wildcard at closest encloser *.a.example.org. for any type other than MX or CNAME. NSEC5 RRs are precomputed.

```
baproh.example.org. 86400 IN NSEC5      48566 1 JQBMG4 A TXT RRSIG
baproh.example.org. 86400 IN RRSIG      NSEC5 16 3 86400 (
    20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1 )
```

Appendix B. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

pre 00 - initial version of the document submitted to mailing list only

00 - fix NSEC5KEY rollover mechanism, clarify NSEC5PROOF RDATA, clarify inputs and outputs for NSEC5 proof and NSEC5 hash computation.

01 - Add Performance Considerations section.

02 - Add elliptic curve based VRF. Add measurement of response sizes based on empirical data.

03 - Mention precomputed NSEC5PROOF Values in Performance Considerations section.

04 - Edit Rationale, How NSEC5 Works, and Security Consideration sections for clarity. Edit Zone Signing section, adding precomputation of NSEC5PROOFs. Remove RSA-based NSEC5 specification. Rewrite Performance Considerations and Implementation Status sections.

05 - Remove appendix specifying VRFs and add reference to draft-goldbe-vrf. Add Appendix A.

06 - Editorial changes. Minor updates to Section 8.1.

07 - Updated reference to [I-D.irtf-cfrg-vrf], updated VRF ciphersuites.

Authors' Addresses

Jan Vcelak
CZ.NIC
Milesovska 1136/5
Praha 130 00
CZ

EMail: jan.vcelak@nic.cz

Sharon Goldberg
Boston University
111 Cummington St, MCS135
Boston, MA 02215
USA

EMail: goldbe@cs.bu.edu

Dimitrios Papadopoulos
HKUST
Clearwater Bay
Hong Kong

EMail: dipapado@ust.hk

Shumon Huque
Salesforce
2550 Wasser Terr
Herndon, VA 20171
USA

EMail: shuque@gmail.com

David C Lawrence
Dyn
150 Dow Street, Tower Two
Manchester, NH 03101
USA

EMail: tale@dd.org