NSEC5: Provably Preventing DNSSEC Zone Enumeration

Sharon Goldberg*, Moni Naor†, Dimitrios Papadopoulos*
Leonid Reyzin*, Sachin Vasant*, Asaf Ziv†

*Boston University
†Weizmann Institute

Posted July 25, 2014; Updated October 17, 2014.

Abstract

DNSSEC is designed to prevent network attackers from tampering with domain name system (DNS) messages. The cryptographic machinery used in DNSSEC, however, also creates a new vulnerability—*zone enumeration*, where an adversary launches a small number of online DNSSEC queries and then uses offline dictionary attacks to learn which domain names are present or absent in a DNS zone.

We propose a new cryptographic construction that solves the problem of DNSSEC zone enumeration while remaining faithful to the operational realities of DNSSEC. NSEC5 can be thought of as a variant of NSEC3, where deterministic RSA-based *keyed* hashing scheme is used instead of an unkeyed hash function. With NSEC5, nameservers hold an RSA hashing key and perform an online public-key signature for each query. However, unlike existing DNSSEC online signing solutions (RFC 4470), leaking this RSA hashing key can only compromise protection against zone enumeration, but does *not* harm the integrity of the zone in any way. Put another way, leaking the RSA hashing key only downgrades the security of NSEC5 back to that of NSEC3.

Moreover, we prove that online public-key operations are necessary to prevent zone enumeration. Specifically, we prove that security against network attackers and privacy against zone enumeration cannot be satisfied simultaneously unless the DNSSEC server performs online public-key cryptographic operations.

1 Zone enumeration issues in DNSSEC

To understand the zone enumeration problem, we can partition the functionalities of DNSSEC into two distinct parts. The first is to provide an authenticated *positive* response to a DNS query. (For example, query: "What is the IP addresses for `www.example.com`?"; answer: "`www.example.com` is at 155.41.24.251.") The second is to provide an authenticated *denial of existence*, when no response to the query is available. (For example, query: "What is the A record for `aWa2j3.example.com`?"; answer: "`aWa2j3.example.com` is a non-existent domain.") DNSSEC deals with these functionalities in different ways.

For positive responses, the authoritative nameserver for the zone keeps a finite set of signed resource records; a record contains, for example, a mapping from one domain name to its IP address(es) and is signed by the zone's secret keys. Importantly, these signatures need not be computed online in response to live DNS queries, but instead can be precomputed ahead of time and stored at the nameserver. This has the twin advantages of (1) reducing the computational load at the nameserver, and (2) eliminating the need to trust the nameserver (since it need not store the

signing key). This second advantage is especially important because most zones have more than one authoritative nameserver, and some nameservers might even be operated by entirely different organizations than the one that administers the zone[1]. In what follows, we will use the term *primary nameserver* (or simply *primary*) to describe nameservers that are trusted, and *secondary nameservers* (or simply *secondary*) to describe those that are not.

The zone enumeration comes into play when we consider DNSSEC negative responses. The trivial idea of responding to every query for a non-existent domain with a precomputed signed NXDOMAIN message opens the system up to replay attacks. The trivial idea of precomputing signed responses of the form "_____ is a non-existent domain" also fails, since the number of possible queries that deserves such a response is infinite, making precomputation of signed responses infeasible. Instead, RFC4034 [4] defined an NSEC record to be used for precomputed denial of existence as follows: a lexicographic ordering of the names present in a zone is prepared, and every consecutive pair of names is signed; each pair of names is an NSEC record. Then, to prove the non-existence of a name (`x.example.com`), the nameserver returns the precomputed NSEC record for the pair of existent names lexicographically before and after the non-existent name (`w.example.com` and `z.example.com`), as well as its associated DNSSEC signatures.[2] While this solution elegantly eliminates the need to trust the nameserver and allows for precomputation, it unfortunately allows for trivial *zone enumeration attacks*; namely, an adversary can use NSEC records to enumerate all the domain names present in the zone.

Why is zone enumeration a problem? This question has created some controversy, with many in the DNSSEC community initially arguing that it is actually *not* a problem (*e.g.,* RFC 4033 [2]), before eventually arriving at consensus that it is a problem from some zones (RFC 5155 [15]). Zone enumeration allows an adversary to learn the IP addresses of all hosts in a zone (including routers and other devices); this information can then be used to launch more complex attacks, some of which are mentioned in RFC 5155:

> Though the NSEC RR meets the requirements for authenticated denial of existence, it introduces a side-effect in that the contents of a zone can be enumerated. This property introduces undesired policy issues. ... An enumerated zone can be used, for example, as a source of probable e-mail addresses for spam, or as a key for multiple WHOIS queries to reveal registrant data that many registries may have legal obligations to protect. Many registries therefore prohibit the copying of their zone data; however, the use of NSEC RRs renders these policies unenforceable.

Indeed, some zones (*e.g.,* `.de`, `.uk`) require protection against zone enumeration in order to comply with European data protection laws [18], [1, pg. 37].

Thus, in 2008, RFC 5155 [15] suggested NSEC3, a precomputed denial of existence technique, designed to make zone enumeration more difficult. With NSEC3, first each domain names present in a zone is cryptographically hashed, and then all the hash values are lexicographically ordered. Every consecutive pair of hashes is an NSEC3 record, and is signed by the authority for the zone. To prove the non-existence of a name, the nameserver returns the precomputed NSEC3 record (and the associated DNSSEC signatures) for the pair of hashes lexicographically before and after the *hash* of the non-existent name.[3]

[1] For example, the zone `umich.edu` has two authoritative nameservers run by the University of Michigan (`dns1.itd.umich.edu` and `dns2.itd.umich.edu`) and one run by the University of Wisconsin (`dns.cs.wisc.edu`) [17].

[2] For simplicity of exposition, we ignore the issues of wildcard records and enclosers in our descriptions of NSEC and NSEC3; see RFC 7129 [10].

[3] There was also a subsequent Internet Draft [9] (that has since expired without becoming an RFC) proposing NSEC4. NSEC4 combines NSEC and NSEC3, allowing zones to opt-out from hashed names to unhashed names. Like NSEC3, NSEC4 is vulnerable to zone enumeration via offline dictionary attacks.

Hashing the names makes trivial enumeration of the zone much more difficult, but the design nevertheless remains vulnerable to zone enumeration using an offline dictionary attack. Specifically, an adversary can issue several queries for random non-existent names, obtain a number of NSEC3 records, and then use rainbow tables (or other dictionary attacks for cracking hashes) to determine the names that are present in the zone from the hashes in the NSEC3 records. In fact, Bernstein's nsec3walker tool [8] does just that, effectively checking up to 2^{34} hash value guesses in one day, using a standard laptop and existing cryptographic libraries.

To blunt the impact of dictionary attacks, the RFCs do introduce a salt value (using the NSEC3PARAM record); however, in contrast to password-hashing applications that mitigate against dictionary attacks by using a unique salt for each user, NSEC3 requires salt to be *common to the entire zone*. Since changing the salt requires re-computing the signatures for the entire zone, RFC 6781 [14] recommends updating the salt only when key-rollover takes place (a very infrequent—monthly, or even yearly—event), which makes the salt a fairly weak defense against dictionary attacks. Moreover, once an adversary has collected a number of NSEC3 records and the salt for the zone, it can use offline dictionary attacks to learn the records present in the zone. Indeed, RFC 5155 acknowledges this:

> The NSEC3 RRs are still susceptible to dictionary attacks (i.e., the attacker retrieves all the NSEC3 RRs, then calculates the hashes of all likely domain names, comparing against the hashes found in the NSEC3 RRs, and thus enumerating the zone).

Our story thus begins here. Today, DNSSEC deployments support NSEC and/or NSEC3 and remain vulnerable to zone enumeration attacks. We use cryptographic lower bounds to explain why zone enumeration attacks could not be addressed by previous designs, and propose a new solution, called NSEC5, that protects against them.

2 Our Cryptographic Model

Our first contribution is the following cryptographic model, which makes precise the desired notion of privacy:

Model. We have a trustworthy source, called a *primary nameserver*, which is trusted to determine the set of names (`www.example.com`) present in the zone and their mapping to corresponding values ("155.41.24.251"). *Secondary nameservers* receive information from the primary nameserver, and respond to DNS queries for the zone. The queries are made by *resolvers*.

Our goal is to design a denial-of-existence mechanism that achieves the following:

(1) Integrity. The primary nameserver is trusted to determine the set of names in the zone and to provide correct responses to DNS queries. However, the secondary nameservers and other network adversaries are not trusted to provide correct responses to DNS queries. The integrity property ensures that bogus responses by secondaries or network adversaries will be detected by the resolver. This is the traditional DNSSEC security requirement of "data integrity and ... origin authentication" described in RFC 3833 [6].

(2) Privacy. Both primary and secondary nameservers are trusted to keep the contents of the zone private. (If they don't, there is nothing we can do, since nameservers necessarily need to know the names present is in the zone.) However, resolvers are not. The privacy property must ensure that the response to a query by a resolver must only reveal information about the queried domain name, and no other names. Our main definitional contribution is the formalization of this requirement to avoid zone enumeration, as was laid out in RFC 5155 [15] and the quote above.

(3) Performance. We would like to limit the online computation that must be done by a nameserver in response to each query. This is discussed in *e.g.,* RFC 4470 [20].

The formal cryptographic model and security definitions are in our technical report [11].

3 Cryptographic Lower Bound

We demonstrate in our technical report [11] that satisfying both the integrity and privacy goals if DNSSEC implies that nameservers must *necessarily* compute a public-key cryptographic signature for each negative response. This explains why the approaches taken by NSEC and NSEC3, which limit the nameserver computation to cryptographic hashes, cannot prevent zone enumeration.

Our technical report also addresses the question of whether our privacy requirements are "too strong" and argues that any meaningful relaxation still implies that responders must perform public-key authentication. Thus we conclude that preventing zone enumeration requires substantial ("public-key") online computation, rather than just private-key computation such as evaluating a cryptographic hash function.

4 NSEC5: A Denial-of-existence Mechanism

Armed with the knowledge that privacy necessitates an online signature computation for every negative response, we present a new solution that requires two online hash computations and a single online RSA computation for each authenticated denial of existence. Our solution, called NSEC5, provably achieves integrity and privacy.

In designing NSEC5, our key observation is that we can "separate" our two security goals (integrity and privacy) using two separate cryptographic keys. To achieve integrity, we follow the traditional approach used in DNSSEC with NSEC and NSEC3, and allow only the primary nameserver to know the primary secret key SK_P for the zone; this primary secret key is used to ensure the integrity of the zone. However, we now make the crucial observation that, while the integrity definition does not allow us to trust the secondary nameserver, our privacy definition does (because if the secondary nameserver is untrusted, then privacy is lost, anyway, since it knows the entire zone). Thus, we achieve privacy by introducing a secondary key SK_S, that we provide to *both* the primary and secondary namesevers. The secondary key is *only* used to prevent zone enumeration by resolvers, and will have no impact on the integrity of the zone. The public keys PK_P and PK_S corresponding to SK_P and SK_S will, naturally, be provided to the resolver, using the standard mechanisms used to transmit public keys and parameters in DNSSEC.

We emphasize that privacy makes sense only when the secondary nameserver can keep some information secret (else, R is no longer private). Thus, the addition of SK_S to the secondary nameserver does not introduce any additional security vulnerability: if it is leaked, integrity is not compromised. SK_S can be distributed to secondary nameservers using the same mechanisms that are used to distribute the zone data.

4.1 The NSEC5 Construction.

Our NSEC5 construction is extremely similar to NSEC3: all we need to do is replace the unkeyed hash used in NSEC3 with a new "keyed hash" F that uses the secondary keys PK_S, SK_S. Our solution is as follows.

The secondary keys $PK_S = (N_S, e_S)$ and $SK_S = (N_S, d_S)$ are an RSA key pair. For each record x that is present in the zone R, the primary nameserver computes a deterministic RSA signature [7]

on x using a hash function h_1

$$S(x) = (h_1(x))^{d_S} \bmod N_S \tag{1}$$

and then hashes it to a shorter string using another hash function h_2

$$F(x) = h_2(S(x)) .$$

The resulting F values are lexicographically ordered, and each pair is signed by the *primary nameserver* using its key SK_P (just like in NSEC and NSEC3). The resulting pair of F values is an NSEC5 record. The primary nameserver then gives the NSEC5 records and the secondary secret key SK_S to the secondary nameserver.

To prove the non-existence of name q queried by the resolver, the *secondary* nameserver computes $S(q)$ and $F(q)$ using SK_S, and responds to the resolver with (1) an NSEC5PROOF record containing the RSA value $S(q)$ and (2) the signed NSEC5 record for the hashes that are lexicographically before and after $F(q)$ ($F(q)$ is the hash of the value in the NSEC5PROOF record). Note that, while the NSEC5 record is precomputed by the primary nameserver, the NSEC5PROOF record is computed online by the secondary nameserver and is *unsigned* (by the primary secret key PK_P).

The resolver can validate the response by (1) using the secondary public key PK_S to verify the RSA signature in the NSEC5PROOF record, checking that

$$(S(q))^{e_S} \bmod N_S = h_1(q)$$

(2) using the primary public key PK_P to confirm that the NSEC5 record is validly signed, and (3) checking that $h_2(S(q))$ (the hash of the RSA value in the NSEC5PROOF record) is lexicographically between the hashes in the NSEC5 record. In other words, $S(q)$ maintains integrity by acting as a "proof" that the value $F(q)$ is the correct "keyed hash" of q.

Note that the keyed hash $F(q)$ must be a deterministic and verifiable function of q. Our specific choice of the deterministic RSA signature algorithm used to compute S in equation (1) is thus crucial; in contrast, any secure signature algorithm can be used to sign the NSEC5 record using SK_P.

4.2 Integrity even if nameservers are hacked & preventing zone enumeration.

Our technical report [11] has a cryptographic proof that our construction satisfies our integrity and privacy definitions. Roughly, privacy follows because the resolver does not know the secondary key SK_S. This eliminates zone enumeration via offline dictionary attacks, since the resolver cannot compute the "keyed hash value" $F(q)$ on its own; the only way it can learn $F(q)$ is by asking online queries to the nameserver (or by breaking RSA!). Meanwhile, integrity follows because only the primary nameserver can sign NSEC5 records; the resolver can use the secondary public key PK_S to verify that the secondary nameserver correctly computed the RSA value $S(q)$ in the NSEC5PROOF, and responded with the right NSEC5 record. If a malicious secondary wanted to announce a bogus non-existence record, (s)he would not be able to produce a properly-signed NSEC5 record covering $F(q)$.

Storing secrets at nameservers. NSEC5 requires secondary nameservers to hold the the *secret* secondary RSA key SK_S. Fortunately, SK_S only needs to be as secure as the records whose the privacy it protects, since leaking SK_S does not compromise integrity in any way. Specifically, if SK_S is leaked or the secondary nameserver becomes adversarial, the integrity of the zone is not compromised; all that is lost is privacy against zone enumeration, effectively downgrading the security of NSEC5 to that of NSEC3. This is in stark contrast to the online-signing solution

proposed in RFC 4470 [20] (and further discussed in RFC 4471 [19]). RFC 4470 suggested that every nameserver (even the secondary) be given the primary key for the zone, and used it to produce online signatures to responses of the form "q is a non-existent domain". With the RFC 4470 approach, integrity is completely lost if the secondary is hacked or its secret key is leaked; meanwhile, NSEC5 preserves integrity even when the secondary nameserver is compromised.

4.3 NSEC5 practical considerations.

The NSEC5 construction allows resolvers to verify using the same technologies they always used: hashing and validation of RSA signatures. NSEC5 does, however, requires a single online RSA computation at the secondary nameserver, making it computationally heavier than NSEC and NSEC3 (and NSEC4). However, our lower bounds prove this extra computation is necessary to eliminate zone enumeration. Additionally, only the zone administrators that require our strong privacy guarantees need to deploy NSEC5; others that don't can just use NSEC or NSEC3. We now compare NSEC5 to the existing DNSSEC standard with NSEC3.

Computational overhead. The main computational overhead of our approach over NSEC3 is online signing at the secondary nameservers. Specifically, we require secondaries to compute a deterministic RSA signature $S(q)$ (sent in the NSEC5PROOF record) *online* for every query q that requires a negative response. Note, however, that online signing has been standardized (RFC 4470 [20]) and implemented in commercial DNSSEC systems [16, Sec. 4], [13].

We also require resolver to verify the RSA signature $S(q)$ in the NSEC5PROOF record. This is no slower than actually verifying the signature on an NSEC record itself, representing no more than an 2x increase in computational overhead. Moreover, comparing this extra overhead to the analogous computation in NSEC3—namely, computing (multiple iterations of) a hash (*e.g.,* SHA-256) on the query q—suggests that NSEC3 and NSEC5 can have *identical* computational overhead at the resolver. Specifically, RFC 5155 [15, Sec 10.3] specifies that the number of iterations in the NSEC3 hash can result in a computation with similar cost as verifying a signature on an NSEC3 record (*e.g.,* 500 SHA1 iterations for a 2048-bit RSA signature).

Finally, the primary also needs to compute some extra signatures when setting up the zone—an additional $r = |R|$ RSA signing operations in addition to the $2r + 1$ signatures needed to sign the NSEC5 records. This represents a 2x increase over NSEC3, which requires $|R| + 1$ signatures on the NSEC3 records.

Transmitting information to the resolvers. We must also consider how the primary nameserver can authentically transmit the secondary public key PK_S and hash functions h_1, h_2 to the resolver. An analogous issue arises in NSEC3, when transmitting the salt and hash function identifier to the resolver; NSEC3 deals with this by including the salt and an "algorithm identifier" for the hash in the NSEC3 record itself, and having the entirety of the NSEC3 record signed using the primary secret key SK_P [15]. We could analogously include the secondary public key PK_S and algorithm identifiers for the hash functions h_1 and h_2 in each NSEC5 record, and then sign the entire NSEC5 record using SK_P.[4] Alternatively, if we want to avoid including PK_S in each NSEC5 record (since an 2048-bit public RSA key is much larger than the 24-bit NSEC3 salt), we could use a separate DNSKEY record, setting the flag bits to indicate it is *not* a signing key for the zone (*e.g.,* using the reserved flag bits, or setting the ZSK and SEP bits to 0, or perhaps by introducing a new DNSKEY flag); this approach complies with RFC 4035 [3]'s discussion of the use of the DNSKEY record for non-zone keys.

[4]Notice that by signing the entire NSEC5 record with SK_P, which is only known to the primary nameserver, we ensure that PK_S, h_1, h_2 are authentically communicated from the primary to the resolver.

Transmitting information to secondary nameservers. Next, observe that the primary resolver must communicate some extra information to the secondary resolvers; namely, the public parameters PK_S, h_1, h_2 and the secondary secret key SK_S. NSEC3 uses the NSEC3PARAM record to transmit the NSEC3 salt and hash function to the secondary resolvers; we could similarly distribute PK_S, h_1, h_2 in an NSEC5PARAM records. Distributing the secondary *secret* keys SK_S may be more cumbersome, however, because secondaries traditionally do not store any secret keys in DNSSEC. Fortunately, since leaking SK_S does not compromise integrity in any way, we could just include SK_S in the NSEC5PARAM record (and require that NSEC5PARAM is never sent to resolvers), or alternatively have the primary directly send SK_S to the secondaries in a TSIG message[5].

Opt-out. Finally, the structural similarity of NSEC5 with NSEC3 and NSEC allows for easy adoption of existing mechanisms such as wildcards, Opt-out (RFC 5155) and Opt-in (RFC 4956 [5]). We will address these complications in future versions of this work; here we only note that several space-saving proposals from the NSEC4 draft [9] can also be incorporated into our NSEC5 construction.[6]

5 Ongoing work.

We believe NSEC5 presents an attractive alternative to NSEC3 for those zone operators who require strong privacy against zone enumeration. We welcome feedback on this work, and collaboration from practitioners that would like to work with us towards implementing and standardizing NSEC5. Please see our technical report [11] and video [12] for more information about NSEC5.

Acknowledgements

We thank Casey Deccio, Daniel Kahn Gillmor, Ben Laurie, Jared Mauch, Matthijs Mekking, Russ Mundy, Benno Overeinder, Ondrej Sury, Wouter Wijngaards and the participants of the DNS OARC Fall 2014 Workshop in Los Angeles for useful feedback and suggestions. This material is based upon work supported by the US National Science Foundation under Grants 017907, 1347525, 1012798, and 1012910, the Israel Science Foundation , BSF and IMOS, and from the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Brian Aitken. Interconnect communication MC / 080:DNSSEC Deployment Study. `http://stakeholders.ofcom.org.uk/binaries/internet/domain-name-security.pdf`, 2011.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, Internet Engineering Task Force, March 2005.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035, Internet Engineering Task Force, March 2005.

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

[5] TSIG is used by the primaries to authentically and dynamically update information stored at the secondaries.

[6] Using the wildcard optimization from NSEC4 [9], our denial-of-existence response (containing two NSEC5 records and two NSEC5PROOF records in the worst case) is only about one RSA-value (*e.g.,* 2048 bits) longer than today's unoptimized NSEC3 standard (containing three NSEC3 records in the worst case).

[5] R. Arends, M. Kosters, and D. Blacka. DNS Security (DNSSEC) Opt-In. RFC 4956, Internet Engineering Task Force, July 2007.

[6] D. Atkins and R. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, Internet Engineering Task Force, August 2004.

[7] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[8] Daniel J. Bernstein. Nsec3 walker. `http://dnscurve.org/nsec3walker.html`, 2011.

[9] R. Gieben and W. Mekking. DNS Security (DNSSEC) Authenticated Denial of Existence. IETF DNSEXT Internet Draft `http://tools.ietf.org/html/draft-gieben-nsec4-00`, January 2012.

[10] R. Gieben and W. Mekking. Authenticated Denial of Existence in the DNS. RFC 7129, Internet Engineering Task Force, February 2014.

[11] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration. Technical report, Cryptology ePrint Archive, Report 2014/582, 2014. `http://eprint.iacr.org/2014/582`.

[12] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration, October, 2014. YouTube: `http://youtu.be/t4tro7BP6CA`.

[13] Dan Kaminsky. Phreebird. `http://dankaminsky.com/phreebird/`, 2011.

[14] O. Kolkman, W. Mekking, and R. Gieben. DNSSEC Operational Practices, Version 2. RFC 6781, Internet Engineering Task Force, December 2012.

[15] B. Laurie, G. Sisson, R. Arends, and D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155, Internet Engineering Task Force, March 2008.

[16] PowerDNS. *PowerDNS manual*, December 2013.

[17] Venugopalan Ramasubramanian and Emin Gün Sirer. Perils of transitive trust in the domain name system. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 35–35. USENIX Association, 2005.

[18] Marcos Sanz. Dnssec and the zone enumeration. European Internet Forum: `http://www.denic.de/fileadmin/public/events/DNSSEC_testbed/zone-enumeration.pdf`, October 2004.

[19] G. Sisson and B. Laurie. Derivation of DNS Name Predecessor and Successor. RFC 4471, Internet Engineering Task Force, September 2006.

[20] S. Weiler and J. Ihren. Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC 4470, Internet Engineering Task Force, April 2006.