Stretching NSEC3 to the Limit: Efficient Zone Enumeration Attacks on NSEC3 Variants

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

February 8, 2015

Abstract

We present efficient zone enumeration attacks against variants of DNSSEC with NSEC3 that do not use online signing.

1 Introduction

Recently, in a paper proposing NSEC5, a denial of existence mechanism for DNSSEC [3], we also proved that security against (1) network attackers that tamper with DNS messages and (2) privacy against zone enumeration cannot be satisfied simultaneously, unless the DNSSEC nameserver performs online public-key cryptographic operations for each denial-of-existence response it sends. This explains why NSEC3 with precomputed responses is vulnerable to zone enumeration; indeed, zone enumeration attacks have been demonstrated by Bernstein [2] and Wander et al. [6]. More importantly, our results in [3] show that *any* hash-based denial of existence scheme that provides integrity against network attackers will still be vulnerable to zone enumeration.

Nevertheless, one might still be tempted to attempt to modify NSEC3 using only hash-based techniques, in order to make zone enumeration more difficult. We emphasize that our proof in [3] shows that schemes that do not use online signing will still be vulnerable to zone enumeration.[1] However, in this report we will demonstrate efficient zone enumeration attacks

*Boston University, Department of Computer Science. Email: {goldbe,dipapado,reyzin,sachinv}@cs.bu.edu. Research supported in part by the US National Science Foundation under grants 1017907, 1347525, 1012798, and 1012910 and by a gift from Verisign Labs.

†Weizmann Institute of Science, Department of Computer Science and Applied Mathematics. Email:{moni.naor,asaf.ziv}@weizmann.ac.il. Incumbent of the Judith Kleeman Professorial Chair. Research supported in part by grants from the Israel Science Foundation , BSF and IMOS and from the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation.

[1]Note that "NSEC3 White Lies" does use online signing, and thus is NOT vulnerable to zone enumeration. However, with NSEC3 White Lies, each nameserver must hold the zone-signing key for the zone, and a compromise of the nameserver completely compromises the integrity of the zone. Meanwhile, NSEC5 also

against two schemes that have been proposed to us, as alternatives to "vanilla" NSEC3. We start by recalling vanilla NSEC3, and then describe its two variants.

Vanilla NSEC3. With NSEC3, each domain name 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.

NSEC3 with dummy records. We add dummy hash values to the NSEC3 chain, to force an attacker to spend additional effort to enumerate the original zone, as follows:

1. Take all names in the zone and hash them to obtain a list of R hashed names; $\Lambda = \{h_1, h_2 \dots h_R\}$.
2. Pick m random numbers $r_1, \dots, r_m$ and append them to Λ.
3. Lexicographically sort Λ.
4. Sign every adjacent pair in the list; each pair is an NSEC3 record.

Subsequently, queries are answered in the same way as with NSEC3, with the server responding with the unique "covering" NSEC3 record for the queried name. Note that, some of the records have dummy random numbers (for one or both of their boundaries), rather than actual names as in vanilla NSEC3.

Noisy NSEC3. This approach can be seen as a modification of the NSEC3 with dummy records scheme, where each hash value of an existing name is "perturbed" by a small amount of noise. The scheme is parameterized by an integer b, the "noise amplitude" or upper-bound on the bit-length of the noise. Typically, we bound b by the size of the output of the hash algorithm. So, for SHA-256, we have $0 \leq b \leq 256$.

1. Take all names in the zone and hash them to obtain the list of R hashed names. Let the list be $\Lambda = \{h_1, h_2, \dots h_R\}$.
2. For each hashed name h_i, choose two random b-bit numbers a_i and c_i and compute the two "noisy hash" values $h_i + a_i$ and $h_i - c_i$. Add these noisy hash values to Λ.
3. Sort Λ lexicographically.
4. Sign every adjacent pair in the list; each pair is an NSEC3 record.

Queries are again handled in the exact same manner as before. The final list is of size $3R$.

Overview of this report. Our goal is to show efficient zone enumeration attacks against NSEC3 with dummy records and against Noisy NSEC3. We start in Section 2.1 by presenting the classic zone enumeration attack used by Bernstein [2] and Wander et al. [6], and discussing

uses online signing, but a compromise of the nameserver does not harm integrity, because signatures are computed using the the NSEC5KEY, rather than the zone-signing key. If the NSEC5KEY is leaked, then zone enumeration is possible but violations of integrity are not (just like with NSEC3).

the efficiency of the attack. We then propose our own modified version of this attack in Section 2.2, that will be useful for the zone enumeration attacks attacks against the NSEC3 variants. In Sections 3 and 4 we discuss the modified versions of NSEC3, together with efficient attacks against both of them. We conclude in Section 5 with a discussion of our findings and a comparison with NSEC5.

2 Zone enumeration attacks

We start by describing the classic zone enumeration attack used on vanilla NSEC3, as well as our own modified zone enumeration attack that will be useful later when we attack the variants of NSEC3.

2.1 The classic zone enumeration attack

The classic zone enumeration attack has two phases: online and offline. In the online phase, the attacker first collects the *entire NSEC3 chain*, all the hashes of all the names in the zone; this was first proposed by Bernstein [2] and implemented in the nsec3walker mechanism. In the offline phase, the attacker "reverses" all the hashes it collects; typically, the offline phase is a dictionary attack, but other approaches can also be used. Recently, Wander et al. [6] performed this classic zone enumeration attack against the `.com` zone; their offline phase involved building a dictionary using combinations of words frequently used for naming Internet resources and common multi-grams of 1 to 15 characters (including numerals). Wander et al. [6] also tried also brute-force guessing and Markov chains to invert hash values, their results clearly indicate the very limited effectiveness of these attacks, when compared to the the dictionary attack. As such, we shall concentrate on offline dictionary attacks in this report.

More formally, the classic attack is as follows:

Online phase : Retrieve all NSEC3 records (h, h') from the zone.

1. Initiate an empty list of NSEC3 records $\mathcal{L}$
2. Generate a candidate domain name x
3. If $h(x)$ is not covered by any record of $\mathcal{L}$, issue DNS query for x and insert the received NSEC3 record to $\mathcal{L}$
4. If the NSEC3 chain is not complete, go to step 1

Offline phase : Compute the hash of all names in the dictionary D and identify the matching values from the received NSEC3 records.

1. Initiate an empty list of domain names $\mathcal{N}$
2. For each value $x \in D$:
   (a) Compute $h(x)$

(b) For each record $(h, h') \in \mathcal{L}$: if $h(x) = h$ or $h(x) = h'$, then add x to $\mathcal{N}$

3. Return $\mathcal{N}$

At the end of the above process, $\mathcal{N}$ will hold the attacker's view of the zone; all names in $\mathcal{N}$ are in the zone, unless a collision for the hash function has been found. If the zone has size R and the dictionary D contains aR names from the zone for $0 \leq a \leq 1$, then $|\mathcal{N}| = aR$.

Remark on attack formulation. Here we set the terminating condition for the online phase to be the completion of the NSEC3 chain, *i.e.,* retrieving all the NSEC3 records in the zone. In practice, however, as more NSEC3 records are retrieved, "hitting" the last few NSEC3 records will become much harder. Hence, a relaxed terminating condition may be used, where the attacker has an estimate R' of the actual size R of the zone and stops after that many records have been retrieved.

Attack efficiency. We will measure the efficiency of the attack in terms of two very important quantities: the *number of DNS queries in the online phase*, and the *total number of hash computations*.

Online DNS queries. There will always be exactly R online queries, since a query is generated only if it yields a new NSEC3 record[2], and the attack terminates when all records are collected.

Offline hash computations. Hash computations will occur during both the online and offline phases. During the online phase, the process of randomly generating values and hashing them to hit NSEC3 records, can be approximated (if we assume the hash function is a random oracle) by the well-studied coupon collector's problem [5] which yields an expected number of $R \cdot H_R$ random values, where H_i is the i-th harmonic number.[3] Asymptotically, this process takes $O(R \log R)$ hash guesses. During the offline phase, there is one hash computation per value in the zone, which brings the expected total number of queries to $R \cdot H_R + |D|$.

Efficiency in practice. How fast is this zone-enumeration attack in practice? In particular we are interested in the time to do a hash computation. Given the computational capabilities of modern GPU processors, the authors of [6] computed 2.5 billion hashes for names in approximately 200 seconds, using SHA-1 with 140 iterations on a standard desktop with an AMD HD 7970 GPU.

For the online phase, the average number of trials to generate a domain name that falls within a the bounds of a particular NSEC3 record is inversely proportional to the "size" of the NSEC3 record, *i.e.,* $|h - h'|$ where h and h' are the hash values in the NSEC3 records. If a record covers 10% of the domain-space, then one may obtain a conforming domain-name in an average of 10 trials. Note that, the expected NSEC3 record size with SHA-265 will be $2^{256}/R$. In particular, guessing a name that yields a new NSEC3 record took on average 2059 hashing attempts (and approximately *less than* $1ms$) in [6].

[2]Unless one of the generated random names happens to be also in the zone, in which case the DNS response is a positive one; this can happen with negligible probability in practice.

[3]In this analysis, we assume that if we have R NSEC3 records in the zone, then each of the NSEC3 records is of equal "size", where the size of an NSEC3 record with hash values h, h' is $|h - h'|$. In fact, the *expected value* of the size of the NSEC3 records will be equal, but not the values themselves. We will be using this expected value analysis throughout this report, for simplicity of presentation.

In the offline phase, the work is dominated by computing the hash of each value in the dictionary, with the salt value and the number of iterations specified by the zone's configuration[4]. Afterwards, the success of the attack (i.e., the overall number of correctly enumerated names) depends on the "quality" of the dictionary. In [6], the authors put together their dictionary, using popular website statistic trackers, and managed to match approximately 62% of the retrieved hashed names, after *only 14 hours of computation*.

This attack is clearly very successful, under any reasonable metric. It required a commodity desktop with a GPU of moderate cost (approx. $400) and a little more than 1 day to successfully retrieve more than 62% of the names in the `.com` zone[5].

2.2 A dictionary-based zone-enumeration attack

Here we present a modification of the classic zone enumeration attack that achieves the same success rate, but will be more efficient for the modified NSEC3 schemes we will discuss next.

In this attack, we will *not* collect the entire NSEC3 chain of all the NSEC3 records in the zone. Why not? To see why, suppose that the process for "reversing" hashes is entirely based on a dictionary attack, as suggested by the results of Wander et. al [6]. It follows that if a name x is not in the dictionary, then retrieving the NSEC3 record that has $h(x)$ as a (upper or lower) bound is of no help to the adversary. Thus, in our modified attack, the attacker will not bother obtaining the NSEC3 records for hashes of names that are *not* in his dictionary. Instead, the attacker will collect as many distinct NSEC3 records as possible using only names from the dictionary D, as follows:

1. Initiate an empty list of NSEC3 records $\mathcal{L}$ and an empty list of domain names $\mathcal{N}$
2. For each value $x \in D$:
   (a) Compute $h(x)$
   (b) If $h(x)$ is not contained in any record of $\mathcal{L}$, issue DNS query for x
      i. If the response is negative, add the received NSEC3 record to $\mathcal{L}$
      ii. Else add x to $\mathcal{N}$
   (c) For each record $(h, h') \in \mathcal{L}$: if $h(x) = h$ or $h(x) = h'$, then add x to $\mathcal{N}$ if it is not already present in $\mathcal{N}$
3. Return $\mathcal{N}$

The attack achieves the same success rate as the classic attack, *i.e.,* it successfully retrieves aR records from the zone if D contains aR names from the zone for $0 \leq a \leq 1$.

To see why this is the case, observe that if a name x is in the zone then it is not covered by any NSEC3 record, and it appears as a bound of exactly two NSEC3 records. Therefore, if x appears also in D, there are two cases: either it will trigger a new DNS query with

[4]The `.com` TLD employs only one hash iteration and no salt value, as of December 2014.

[5]The entire attack also included guessing hash pre-images with two other techniques, brute-force guessing and Markov attacks, that took an additional 3-4 days but only yielded an additional 2% of names, for a total of 64% in under 5 days.

positive response and therefore it will be added to $\mathcal{N}$ (step 2.(a).ii above), or it will be the bound of an NSEC3 record collected as part of negative response (step 2.(a).i above) and later identified and added to $\mathcal{N}$ (step 2.(c) above). These two cases are not necessarily non-overlapping –e.g., x may be both retrieved with a positive response and as the bound of an NSEC3 record of a negative response– but at least one of them must occut, hence either way, if x is also in D it will eventually end up in $\mathcal{N}$.

Efficiency. We again measure the number of DNS queries and the number of hash computations required.

Online DNS queries. Oserve that an NSEC3 record (h, h') will be retrieved *only* if there exists $x \in D$ with $h < x < h'$. In the best case for the attacker, this will be true for all R NSEC3 records in zone, hence at most R queries. However, because we are going with a dictionary-based attack, the following "bad event" could occur: if a name $x \in D$ is queried, the response will yield a positive response instead of an NSEC3 record, and afterwards, a different name $x' \in D$ would be queried such that the NSEC3 record retrieved has $h(x)$ as its bound. Observe that in the above example, the bad event only happens if x is queried before x'. In the worst case this bad event would happen for all the aR names in the dictionary that also exist in the zone. Therefore, summing up the DNS queries with negative and positive answers, the total upper bound is $R + aR = (a + 1)R \leq 2R$ DNS queries. For comparison, the previous attack took exactly R queries. Hence, with this attack there is a small increase on the upper bound of necessary DNS queries.

Offline hash computations. The number of hash computations is exactly $|D|$. Recall that the previous attack required an expected number of $R \cdot H_R + |D|$ hash computations, which implies a significant improvement in offline computation with our modified attack.

3 NSEC3 with dummy records

As described in Section 1, NSEC3 with dummy records involves adding m dummy hash values to the NSEC3 chain, thus forcing the attacker has to expend additional effort to collect all the NSEC3 records and enumerate the zone.

3.1 The dictionary-based attack on NSEC3 with dummy records.

The attack of Section 2.2 can be successfully applied to this scheme, and its success and efficiency is analyzed below. For simplicity of presentation we assume that there are no names in the dictionary D whose hash collides with one of the dummy values $r_1, \ldots, r_m$. This is a reasonable assumption since these collisions can only happen with negligible probability in practice because (assuming SHA-256 is used for hashing) these r_i are chosen at random from $\{0,1\}^{256}$ and the size of the dictionary $|D|$ is much smaller. (In [6] there were less than 2^{44} names in the dictionary.)

First, observe that aR names from the zone are successfully retrieved. This follows directly from our analysis of the attack in Section 2.2. Regarding efficiency, we measure the number of DNS queries and hash computations.

Online DNS queries. There are a total of $R + m$ NSEC3 records, where m is the number of dummy values and R is the size of the zone. As such, the online-phase of the dictionary-

based attack requires some additional DNS queries compared to that against vanilla NSEC3. The number of queries with negative quries (*i.e.,* those that retrieve NSEC3 records) will be at most $R + m$, since each query will retrieve a previously unseen record and, at best, all of them will be collected. By our analysis in Section 2.2, there can also be at most aR additional queries with positive answers. It follows that the total number of online DNS queries is at most $R + m + aR = (a + 1)R + m \leq 2R + m$.

Offline hash computation. Because of the structure of the attack, the number of hashes is always fixed to $|D|$, *i.e.,* one hash per value in the dictionary.

The small increase in the number of DNS queries performed by the attacker is only natural; the nameserver effectively increases the zone size, hence there are more records to collect. For example, if the number of dummy records is $m = (K - 2)R$, the attack will require at most $2R + (K - 2)R = K \cdot R$ DNS queries. Observe that the attack on vanilla NSEC3 took at most $2R$ queries. Here, the nameserver had to do roughly K times more work during setup (and pay corresponding storage) to make the attack only $K/2$ times more costly for the attacker.

3.2 The classic attack on NSEC3 with dummy records.

Alternatively, if the attacker wishes to collect all the NSEC3 records in the zone, it can launch the classic attack from Section 2.1. From the analysis of the coupon-collector problem, the expected number of hash computations is $(R + m)H_{R+m} + |D|$. The number of DNS queries will always be $R + m$, i.e., one for each NSEC3 record in the zone. If, for example, $m = (K - 1)R$ then the NSEC3 chain length is $K \cdot R$, and the attack will take $K \cdot R$ DNS queries and $O(K \cdot R \log(K \cdot R) + |D|)$ computations. Overall, we have the same trade-off, with the nameserver doing roughly K times more work and the attack becoming less than K times costlier.

4 Noisy NSEC3

An alternative way to make zone enumeration harder is to add parametrized noise to existing NSEC3 records. This approach can be seen as a modification of the previous scheme, where each hash value of an existing name is "perturbed" by a small amount of noise (instead of introducing fresh random hash values). The scheme is, described in Section 1, is parameterized by a integer b, where b is the "noise amplitude" or upper-bound on the bit-length of the noise. Typically, we bound b by the size of the output of the hash algorithm; for SHA-256, we have $0 \leq b \leq 256$.

Before we attack Noisy NSEC3, we must clarify that simply adding noise to the hashed names is not secure. To see why, assume that q_0, q_1 are two names in the zone with corresponding hashes h_0, h_1 in a plain NSEC3 scheme, that are consecutive after sorting. Then one might produce a perturbed version of the NSEC3 chain by adding random noises a_0, a_1 resulting in a list containing the NSEC3 record: $(h_0 + a_0, h_1 + a_1)$ that is consequently signed with signature σ_0. One might imagine 'noising' every NSEC3 record in this manner, in order to make zone enumeration more difficult for the adversary. However, this is not secure. An adversarial nameserver, or one that is compromised, can produce the response

$[(h_0 + a_0, h_1 + a_1), \sigma_0]$ as proof for the non-existence of the queried name q_1; the resolver will correctly validate the signature σ_0 and the answer as $h_0 + a_0 < h(q_1) < h_1 + a_1$ and falsely conclude that the name q_1 is not in the zone.

Therefore, Noisy NSEC3 must introduce *additional* hash values around the existing hash values in the NSEC3 chain. Since we want the additional hash values to not cover existing hashed names (due to the security compromise discussed above), the hashes of the original names must also be appear in the NSEC3 chain. Noisy NSEC3 achieves this by both adding and subtracting some small noise to each hashed name to compute additional hash values, and then proceeds with normal NSEC3 setup, as described in Section 1.

The high-level idea behind Noisy NSEC3 is that, if the noise amplitude b is relatively small, it will be hard for the attacker to hit one of the two NSEC3 records (one above and one below) that contain the hash of an existing name.[6] While this NSEC3 variant originally suggested with *small* noise amplitude b in mind, we evaluate attacks for all possible with noise lengths, *e.g.,* for $b \in [0, 256]$ when SHA-256 is used for hashing.

Next we evaluate our attack when performed against NSEC3. In Appendix A we present a more complicated and less efficient attack that aims at collecting the entire NSEC3 chain.

4.1 The dictionary-based attack on Noisy NSEC3

Our dictionary-based attack is again very successful on Noisy NSEC3. In fact, the analysis is the same as for the case of NSEC3 with dummy records and all aR names from the zone that appear in D will be retrieved.

The hash computations are once again exactly $|D|$, *i.e.,* , one hash per name in the dictionary. Regarding the necessary number of DNS queries, the same argument as with NSEC3 with dummy records applies. There exist $3R$ NSEC3 records, and at most all of them may be collected, each with a fresh query. Therefore there will at most $3R$ queries with negative responses. On top of that, up to aR additional queries are required, for cases where a query is answered positively (*i.e.,* , it returns a name from the zone but not an NSEC3 record), for a total upper bound of $3R + aR \leq 4R$ DNS queries.

5 Discussion

In this technical report, we presented two modified versions of NSEC3 that attempt to make zone enumeration harder by introducing randomized record values. We also, presented a modified version of the attack technique of [2] and [6], that makes use of the existing dictionary (already assumed by the above attacks) during the NSEC3 record collection phase, and drastically reduces the number of necessary hash computations.

As demonstrated by our analysis, our attack is very efficient against both presented schemes and achieves the same level of success, *i.e.,* it successfully retrieves the same number

[6] Another way to think about this NSEC3 modification, is that it behaves in a similar manner as NSEC3 white lies (introduced in [4]) but the noise is introduced around the hashes of the existing names (as opposed to the hash of the queried name), and without online record creation and signing; all ad-hoc boundaries are pre-computed and pre-signed.

of names from the zone. The only trade-off is that it does not retrieve the entire NSEC3 zone, but only those names that are matched in the dictionary.

Using our attack, we also showed that both schemes can make zone enumeration attacks costlier for the attacker by only a linear factor, while at the same time imposing the same additional cost to the nameserver. At a high level, honest nameservers have to pay K times more cost for zone signing and record storing in order to make the attack less than K times harder for the attacker. This is a poor security-efficiency trade-off, made even worse by the following practical observation. Doubling the zone size (via either scheme) would cause the attacker to perform an attack against a zone of twice as large. Naively, this means that the attack takes twice as long. In practice however, the attack can be launched in the same amount of time, by simply using two GPU's for hashing instead of one.

Of course, the trade-off entails costs of a different nature: the honest parties perform more hash operations once (during setup) and then simply store additional records, while the attacker needs to increase his computational cost. A similar effect for the nameserver can be achieved by more trivial techniques, e.g., by multiplying the number of iterations for hash computations by K, or by maintaining K separate copies of the NSEC3 chain, each using a different salt value. However, this linear trade-off is a stark contradiction to the exponential trade-off that occurs in cryptographic schemes, e.g., when the length of cryptographic keys increases. In the latter situation, increasing the cost for honest parties only slightly (e.g., by increasing the key-length by one bit), doubles the expected necessary computational effort for the attacker.

The fact that neither of these schemes truly addresses the problem of zone enumeration comes as no surprise. In our paper [3] we prove that any DNSSEC scheme that wishes to simultaneously achieve security against attackers that may tamper with DNS messages (including secondary nameservers) and privacy against zone enumeration, cannot be based on pre-computed signatures and hashing alone; it must entail online public-key cryptographic operations.

Compared to these schemes, NSEC5 makes zone enumeration exponentially harder (guessing whether a name is in the zone, without explicitly issuing a DNS query for it, is approximately as hard as forging an RSA signature) by introducing only a constant multiplicative overhead for the nameservers. As explained in more detail in our paper [3], this additional cost comes from replacing the iterated hash computation in NSEC3 by a single RSA operation.

Acknowledgements

We thank Justin Cappos, Paul Hoffman, Samuel Weiler and various attendees of IETF90, IETF91 and the DNS OARC Fall 2014 workshop for raising the questions discussed in this technical report.

References

[1] Petra Berenbrink and Thomas Sauerwald. The weighted coupon collector's problem and applications. In *Computing and Combinatorics, 15th Annual International Conference,*

COCOON 2009, Niagara Falls, NY, USA, July 13-15, 2009, Proceedings, pages 449–458, 2009.

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

[3] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration. *IACR Cryptology ePrint Archive*, 2014:582, 2014.

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

[5] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[6] Matthäus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis. GPU-Based NSEC3 Hash Breaking. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 137–144, Aug 2014.

A Recovering all NSEC3 records against Noisy NSEC3

We already presented a very efficient attack against NSEC3 that retrieves all names from the zone that also appear in D.

If the attacker is hellbent on recovering all the NSEC3 records in the zone, then Noisy NSEC3 can indeed make its life more difficult[7]. Nevertheless, for completeness, we consider now an attack whose goal is to collect the entire NSEC3 chain. Noisy NSEC3 does indeed make the collection of the entire NSEC3 chain more difficult, but still possible.

Intuitively, if b is small, it will be hard to come up with a random hash pre-image that hits a record of the form $h_i + a_i$ or $h_i - c_i$; however, we shall show a new attack that successfully retrieves all hashed names from the zone when b is small.

The attack is based on this observation: If b is sufficiently large, then the list Λ produced during setup will simply look like NSEC3 with the dummy records, for which we already showed an efficient attack.

It remains to show what happens when b is small. In this case we distinguish two types of records: *large* NSEC3 records, of the form $(h_0 + a_0, h_1 - c_1)$ and *small* NSEC3 records, of the form $(h_0, h_0 + a_0)$ or $(h_0 - c_0, h_0)$. Then, hitting the "useful" small records (that contain hashes of names as bounds) will indeed be hard. However, the crucial observation is that in this case hitting the large ones will be easy, and that is what our attack utilizes.

This attack is parameterized by a threshold value between 0 and 256 (intuitively for values below the threshold, the noise is perceived as small) that and proceeds as follows:

1. If $b >$ *threshold* perform the attack of Section 2.1. Else proceed as follows:

[7]However, as we argued above in Section 2.2, there may not be much sense in this sort of attack, since if a name x is not in the dictionary, then retrieving the NSEC3 record that has $h(x)$ as a (upper or lower) bound is of no help to the adversary, unless he dedicates resources to inverting $h(x)$.

2. **First online phase.**

   (a) Initiate an empty list of NSEC3 records $\mathcal{L}$.

   (b) Generate a candidate domain name x.

   (c) If $h(x)$ is not covered by any record of $\mathcal{L}$, issue DNS query for x and insert the received NSEC3 record to $\mathcal{L}$.

   (d) If your have not collected all the large records, go to Step (b)

3. **First offline phase.**

   (a) For each name $x \in D$, compute hash $h(x)$

   (b) Sort the records in $\mathcal{L}$ and let (h_i, h'_i) be the i-th record after the sorting.

   (c) Initiate R empty lists $\mathcal{L}_1, \ldots, \mathcal{L}_R$.

   (d) For each name $x \in D$, if $h'_i \leq h(x) \leq h_{i+1}$ add d to $\mathcal{L}_i$.

   (e) Initiate an empty list of names $\mathcal{N}$.

4. **Second online phase.**

   (a) Initiate an empty list of NSEC3 records $\mathcal{L}'$.

   (b) For $i = 1, \ldots, R$ choose at random a name x from $\mathcal{L}_i$, issue corresponding DNS query and append the received NSEC3 record to $\mathcal{L}'$ if the response is negative. If the response is positive, insert x to $\mathcal{N}$.

5. **Second offline phase.**

   (a) For $i = 1, \ldots, R$ compare the hashed values of names in $\mathcal{L}_i$ with the hash value of the i-th entry of $\mathcal{L}'$. If a name matches, insert it to $\mathcal{N}$.

6. Return $\mathcal{N}$.

By following the online phase of the attack from Section 2.1, the attacker can retrieve all the large records without much additional effort and the first online phase terminates when this happens. Then, the attacker ignores all those names from D that are covered by the retrieved records (for security purposes these cannot correspond to existing names, as discussed above). All the values that remain in the dictionary are candidate names. The attacker proceeds to group the dictionary values in buckets based on which "gap" between the collected NSEC3 records that the dictionary values fall into. Assuming all large records have been retrieved in the online phase, each such gap contains one name that is in the zone. To retrieve the name in the "gap", the attacker performs an additional online phase where he asks one DNS query per "gap" using a randomly-chosen name for the part of his dictionary that corresponds to that gap. The response can either be positive (if the attacker happened to choose the correct name), or it will contain an NSEC3 that will contain the hash of the name that it in the zone as its upper or lower bound. Finally, in a second offline phase, the attacker performs a pass over each bucket to identify which value matches the one in the

response; this process looks like the offline phase of the classic attack from Section 2.1 but with many tiny dictionaries (one for each "gap") instead of a single large dictionary.

We stress that the idea of small and large records is only used to facilitate the analysis of the attack. In practice, if b is significantly large, it is quite possible to have records that are of neither type. To better understand this, assume a noise value a_i is such that $a_i > h_{i+1} - h_i$. The produced records (in order) will be $(h_i - c_{i+1}, h_{i+1}); (h_{i+1}, h_i + a_i); (h_i + a_i, h_{i+1} + a_{i+1})$. We call this (and other similar cases) an *overlap* and to simplify the analysis we will assume, when appropriate, that overlaps do not occur.

Efficiency and success of the attack. Somewhat surprisingly, our analysis will show that, although Noisy NSEC3 was suggested with small noise values in mind, it performs better for some intermediate values of b. That is, its performance depends on how the threshold is chosen.

What the threshold value corresponds to is the attacker's confidence that he can retrieve *all* records by simply generating names at random. If the noise b is below the threshold, then the attacker believes that the "small records" are too difficult to collect. If the noise is above the threshold, then the attacker believes collection of the full NSEC3 chain is easy. Therefore the choice of threshold is crucial for the efficiency of the attack. In what follows, we analyze how this choice of threshold should be done.

In order to appropriately choose the threshold level, we define value C, that will help us partition the noise levels. It is defined as

$$C = \frac{\sum_{1}^{2R} \text{size of small record}}{\text{size of large record}} = \frac{2R \cdot 2^{b-1}}{2^u/R - 2^b} = \frac{2^{b+r}}{2^{u-r} - 2^b}$$

where $r = \log R$, and 2^u is the range of the hash function, i.e., $u = 256$ if SHA-256 is used.

C is a measure of the relative size of the introduced records around the hash of a name, and a record between two noisy values. For small values of b, $C \ll 1$, and C grows as b increases. We will examine what happens if we set the threshold according to various values of C and the corresponding values of noise amplitude b, by analyzing the attacks that can be efficiently launched in each case.

We distinguish the following cases for C:

$\mathbf{C \leq 1}$ This case captures small values of b. It corresponds to scenarios where the sum of the length of all small records is at most as large as the length of one large record. From the above, this can be written as:

$$2^{b+r} + 2^b \leq 2^{u-r} \Leftrightarrow 2^b \leq 2^{u-r}/(2^r + 1) < 2^{u-r}/2^r \Leftrightarrow b < u - 2r$$

Let us now analyze the efficiency of our attack for values of $b < u - 2r$. As usual, we will measure the number of necessary hash computations and DNS queries.

Hash computations. We model the process of random name generation with a modified coupon-collector's problem with $R + 1$ records. The first R coupons are the "large records" and the last one corresponds to *all* of the "small records". That is, whenever the hash of a randomly generated name falls within a small record (which

will happen rarely for these values of b) we suppose that the attack collected only a single coupon. Observe that since that are a total of $3R$ NSEC3 records, collecting a "small record" cannot not cause the number of DNS queries to exceed $3R$. By standard analysis, collecting all coupons (including the R large record) will take $(R+1)H_{R+1}$ hash computations. $|D|$ additional hash computations are performed for the dictionary values, hence asymptotically the total number of hash computations is $O(R \log R + |D|)$, i.e., the same order of hashes as with vanilla NSEC3.

Online DNS queries. During the first online phase at most $3R$ queries may be issued and during the second phase at most R additional queries can be made. However, observe that the upper bound of $3R + R = 4R$ corresponds to a very naive attack that ignores small records retrieved during the first online phase. In practice however, these records contain the actual target information (i.e., hashed zone names) therefore for every such record retrieved, no additional query during the second phase is necessary. This gives a final upper bound of $3R$ queries (same as vanilla-NSEC3 for a zone of size $3R$).

Success of the attack. The attack successfully retrieves aR zone names if the dictionary D contains aR names from the zone. If name y appears both in the zone and the dictionary, and falls inside one of the "gaps" then are there are two cases during the second online phase: either $x = y$, or the returned NSEC3 record has $h(x)$ as upper or lower bound. In both cases, at the end of the attack, it holds that $x \in \mathcal{N}$. Observe that y cannot fall outside all gaps, as this means that there would exist an NSEC3 record that covers it, which would be a security breach.

From the above analysis, for values of $b < u - 2r$ the attack presented here is very efficient in practice. Next we turn our attention to large threshold values.

$\mathbf{C \geq 2R}$ This case captures values of b that are large enough to make the collection of all records feasible. For $C = 2R$ the expected size of a small record is the same as that of a large record. At this noise level, the role of parameterized noise is redundant; it produces the same effect as introducing dummy values into the NSEC3 chain, per Section 3. For $C \geq 2R$ we have:

$$2^{b+r} \geq 2^{u+1} - 2^{b+r+1} \Leftrightarrow 3 \cdot 2^{b+r} \geq 2^{u+1} \Leftrightarrow \log 3 + b + r \geq u + 1 \Leftrightarrow b \geq u - r$$

From our modeling of the attack as a coupon collector's problem, it follows that the expected number of hash computations is $3R \cdot H_{3R} + |D|$, i.e., as many as those necessary against vanilla-NSEC3 for a zone of $3R$ records. Also, the necessary number of DNS queries is at most $(3+a)R \leq 4R$, in order to retrieve aR zone names that appear also in D.

It follows that for $b = u - r$, or any value above that[8], collecting all the hashed names is a viable and efficient option for the adversary.

[8]Intuitively, small records, *i.e.,* the ones that contain the actual hashed names with Noisy NSEC3, become larger than the truly noisy ones, hence easier to hit.

$\mathbf{1 < C < 2R}$ We saw that for values of b below $u - 2r$ or above $u - r$, we have efficient attacks. This leaves the problem of how to handle intermediate noise levels. This noise amplitude can be characterized as $b = u - r - j$ for $j = 1, \ldots, r$, *i.e.,* there are r possible noise levels. At this noise amplitude, the "small records" are small but not small enough that the probability of hitting them is negligible. For example, for $b = u - r - 1$ each introduced noisy record, has an expected size of half the expected size of a large record. That is, in expectation hitting a small record is twice as hard as hitting a large record. More generally, if $j = \delta$, then the expected size of a small record is 2^δ times smaller than that of a large one.

One viable attack would be to proceed for full NSEC3 chain record collection, *i.e.,* to set the threshold for $C = 2R$.

Record collection in this case can be modeled with a weighted coupon collector's problem. As shown in [1], a good approximation (within a sub-logarithmic factor) of the number of necessary trials to collect n coupons is $\sum_{i=1}^{n} 1/(i \cdot p_i)$, where $p_1 \leq \ldots \leq p_n$ are the different probabilities in increasing order (p_i corresponds to the i-the coupon). Let p_l be the probability to hit a large record with a random guess, and likewise p_s for small ones. The above formula in our case yields for the expected number of hash computations (on top of the mandatory $|D|$ hashes for dictionary names):

$$\sum_{i=1}^{2R} 1/ip_s + \sum_{i=2R+1}^{3R} 1/ip_l = (1/p_s)H_{2R} + (1/p_l)(H_{3R} - H_{2R})$$

As a sanity check, observe that when $p_l = p_s = 1/(3R)$ the above formula gives $3R \cdot H_{3R}$, as expected. If $p_l/p_s = 2^\delta$ (*i.e.,* $j = \delta$), substituting above we get:

$$\begin{aligned}
(1/p_s)H_{2R} + (1/(2^\delta p_s))(H_{3R} - H_{2R}) &= 2^{u-b+1}H_{2R} + 2^{-\delta}2^{u-b+1}(H_{3R} - H_{2R}) \\
&= 2^\delta(2R \cdot H_{2R}) + 2R(H_{3R} - H_{2R}) \\
&= 2R(2^\delta \cdot H_{2R} + H_{3R} - H_{2R}) \\
&\approx 2R(2^\delta \cdot H_{2R} + \ln(3R) - \ln(2R)) \\
&= 2R(2^\delta \cdot H_{2R} + \ln 3 - \ln 2) \\
&< 2R(2^\delta \cdot H_{2R} + 1) < 2R(2^\delta \cdot H_{2R} + H_{2R}) \\
&\leq (2^\delta + 1)(2R \cdot H_{2R}) \\
&< (2^\delta + 1)(3R \cdot H_{3R}) < 2^\delta(3R \cdot H_{3R})
\end{aligned}$$

Recall that $3R \cdot H_{3R}$ is the expected number of hash computations to collect all the NSEC3 zones from a zone of size $3R$ employing vanilla NSEC3. At a high level, this implies that proceeding for full record-collection in a scenario where small records are, e.g., K times smaller than the large ones in expectation, takes less than K times more hash computations for the attacker. We already have an efficient attack for the case where $K > 2R$ (that is, $C > 1$ in our previous analysis above). Hence, in the worst case this attack would require less than R times hash computations, than the attack against vanilla-NSEC3 for a zone of size $3R$. On the other hand, the number of online DNS queries is again upper-bound by $3R$.

An alternative attack for these intermediate noise values would be to run the attack collecting the large records (in other words, to set the threshold for $C = 1$). It follows easily that the total number of the DNS queries for such an attack is again upper bounded by $4R$, as in our analysis for $C \leq 1$. The expected number of necessary hash computations can again be estimated by a modification of the coupon collector's problem (similar to the one we made for our analysis for $C \leq 1$). Observe that within these noise levels, the total expected size of small records is upper bounded by twice that of the total expected size of the large ones (which occurs for $C = 2R$). Hence, the collection of large records can be modeled by a game with $3R$ records of the same size (R of which are the large ones and $2R$ the others). It follows that the expected number of hash computations is only $3R \cdot H_{3R} + |D|$. Finally, if all large records are retrieved, it follows that aR names will be retrieved from the zone.

This attack is much more efficient than the previous one. This should come as no surprise as it is a strictly weaker attack that only retrieves R of the $3R$ records. However, measuring the exact effectiveness of this attack is rather complicated. For these noise levels, where the sizes of small and large records are comparable, it is not clear whether large and small records are distinguishable, or even whether R large records exist. This happens because of the possible existence of overlaps (which will realistically occur at these noise levels).

A very rough estimation on the effectiveness of the attack can be made as follows. If the expected size of small records is, e.g., 8 times smaller than that of the large ones, overlaps of noisy values with hashed names should occur in expectation while 1/8 of the noisy values is introduced. Since an overlap can occur either from "above" or from "below" (*i.e.,* , due to a_i or c_i), roughly 1/4 of the time a record will contain an overlap. That is, in expectation 75% of the names will be retrieved correctly at the end of the attack.

Of course, this is a very heavy-handed approximation; it would be very informative to implement this attack against Noisy NSEC3, in order to see how it performs in practice.

The above concludes our analysis for this attack against Noisy NSEC3 and highlights the importance of setting the threshold value appropriately. To provide some concrete numbers for the relation of noise amplitude, zone size and threshold, let us consider the case of SHA-256 for the hash function of choice, and let the zone under attack be the `.com` TLD.

It follows that $u = 256$ and $r = 18$, hence for noise values $b < u - 2r = 220$, collecting the R large records is easy, from our analysis for the first case above. On the other hand, for noise values around $b = u - r = 238$ collecting all $3R$ records is easy, as per our analysis for the second case above. Based on our analysis for the third case, a good candidate for the threshold value might be 3 bits below 238, i.e., 235. From our analysis, if the noise level is above this level, collecting all the records will take in expectation $2^3 = 8$ times more hash computations. The alternative is to have an efficient attack, that will successfully retrieve about $1-2\cdot(1/8)$ of the names that are matched in the dictionary, *i.e.,* $0.75aR$ of the names, if D contains aR of the zone's names.

Again, implementing this attack against Noisy NSEC3 would help shed some light to the

extent in which the hardness of this attack (as well as its success rate) is an artifact of our analysis and modeling.

In practice, we expect a hybrid of the two attack to be most effective, *at all times*: collect as many records as possible, proceed to erase values from D, and then ask queries from names still in D that happen to fall between gaps of the retrieved records.