Network Working Group
Internet Draft

Steven M. Bellovin
AT&T Labs Research

Expiration Date: June 2002

December 2001

Using Bloom Filters for Authenticated Yes/No Answers in the DNS

draft-bellovin-dnsext-bloomfilt-00.txt

1. Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

2. Abstract

Some aspects of DNSSEC, such as NXDOMAIN error messages, require an authenticated answer. Producing this answer requires complex mechanisms, online storage of the zone's secret key, expensive online computations, or massive zone files. As an alternative, we propose storage of authenticated pointers to Bloom filters. This scheme provides large reductions in the size of, and computational expense to produce, partially-signed zone files.

3. Introduction

Some aspects of DNSSEC [RFC2535], such as NXDOMAIN error messages, require an authenticated answer. Producing this answer requires complex mechanisms, online storage of the zone's secret key, expensive online computations, or massive zone files. The current scheme relies on NXT records, which have a number of troublesome properties. Among these are the space required to store and transmit them; additionally, some people have objected that NXT records make it possible to dump a zone by chaining through NXT records.

The expense of storing DNSSEC records for a zone as large as .COM has led some people to suggest an "opt-in" process: only those parties who wish to have a signed record will have one. This raises the question of how to receive an authenticated message saying that a given RRset is not supposed to be signed. Two mechanisms, NOKEY and NXT records, have been proposed; both have their disadvantages.

Both problems could be solved if the DNS server were to digitally sign its answers. But this is too expensive in CPU time (and exposes the server to denial of service attacks), and requires that the signing key be available online, a serious security risk for major zones such as the root and .COM.

We suggest using Bloom filters [Bloom70] to store yes/no answers to such questions. The filter can be precomputed and presigned, but queries to it are quite efficient. The total amount of storage and computation required appear to be significantly less than is needed for today's solutions. Furthermore, it is not possible to recover names from a filter, thus protecting privacy.

One caveat must be mentioned. The filter needed for a zone such as .COM is far too large to ship around in DNS responses. Accordingly, we propose using indirect references to such filters. While this seems to be an inconvenience, in fact using indirection allows load-shifting and load-balancing.

4. Bloom Filters

A Bloom filter is a very efficient way to store information about the existence of a record in a database. It is susceptible to false positives; however, the probability of a false positive can be made as small as desired.

A Bloom filter is an array of m bits, initialized to zero. It requires a set of k hash functions that are independent and produce uniformly distributed output in the range [0,m-1] on the possible inputs.

To add an entry R to the filter, calculate

```
b[1] = H1(R)
b[2] = H2(R)
...
b[k] = Hk(R)
```

and set bits b[i] to 1 in the array.

To see if a record exists, calculate the same b[i] and check the bit values. If all k bits are 1, the record exists; if even a single bit is 0, the record does not exist.

In a database of any reasonable size, it is not possible to determine the input records from the bit array. Many different records can set any one bit; there is no way to tell which records actually did.

If there are n records stored in the Bloom filter, the probability of a false positive is given by

```
P = (1 - (1 - 1/m)**(k*n) )**k
```

which may be approximated by

```
P = (1 - exp(-k*n/m))**k
```

From the equations, it is clear that it is the ratio of the number of records to the bit array size is what is important, rather than the absolute size of either. Further, there is an optimal range for k; values that are too small or too large will produce too many false positives. The exact set of parameters for any filter need to be chosen with some care. Some possible values for use in the DNS are given in Appendix A.

5. A Bloom Filter Resource Record

As mentioned above, a Bloom filter for the DNS will be large: for the .COM zone, at least 75 Mbytes. We clearly do not wish to transmit such bit arrays on a routine basis! Accordingly, the filter RR contains a URL pointing to the actual filter.

In order to query the filter, clients need to know k and m; they also need to know what hash functions were used. We put these values in the RR, with m in units of kilobytes. (Note: our initial value for m will be around 600,000-800,000,000 bits. This is close enough to what can be held in a 32-bit field that we prefer to buy our factor of 8192 in advance.)

As is discussed below, it may be desirable to divide the filter into chunks. It is therefore necessary to include the chunk size in the RR as well.

Open issue: a public key (or certificate containing a public key) is necessary to validate the filter. Where should this key be stored? In a KEY record? In the RR? In some other record? It is not necessary that this key be the same as the one used to sign the zone; in one variant discussed below, it is better that the two not be the same.

Since Bloom filters are specific to any given instance of a zone, the SOA serial number is an essential part of the authentication process. Accordingly, name servers that return a Bloom filter RR SHOULD also return the relevant SOA record in the Additional Information section.

6. Using Bloom Filter RRs

A client who wishes to authenticate an NXDOMAIN response to a secure query first obtains and authenticates a signed Bloom filter record. It then calculates the b[i] values for the desired name, and checks the bit positions in the Bloom filter. Finally, it authenticates the content of the Bloom filter itself. We present two options for how to perform these last two steps.

6.1. Using TLS for Filter Queries

To avoid the need to transmit a large bit array, one option is to query a Bloom filter server via TLS [rfctls]. The client calculates the bit positions, based on the domain name, k, and m, and then queries the URL specified in the filter RR:

https://bloomfilter.ns.example.com?324+3248+23980+89732+...

The server responds with a simple yes/no answer.

To avoid attacks, the client MUST check that the public key in the server's TLS certificate matches that returned by the RR.

Unfortunately, this scheme requires expensive public key operations by the server for each query. Furthermore, it requires that a private signature key be online. Fortunately, there is no need to make this key the same as the zone-signing key. CPU load concerns can be ameliorated by replicating the server, using any standard load-sharing technique. Again, note that in general the contents of the bit array need not be kept confidential.

6.2. Retrieving Filter Chunks

To avoid the need for online, server-based cryptography, we present an alternate scheme based on bit array downloads. For this version, the server divides the bit array into chunks. Each chunk is digitally signed (the signature is calculated over the bit array chunk, the metadata in the filter RR, and the SOA serial number). The chunk size, and hence the number of chunks, is determined by the tradeoff between download size and the number of chunks that must be signed by the server.

A client that connects to the specified URL will receive the chunks that contain the requested bit positions. (Open issue: should the URL contain bit position numbers or chunk numbers? I suspect that the former is better, since it means that both options can be implemented using the same query syntax.)

A client need not query all k values. It can trade off its own need for certainty, up to the limits set by k, against download time. This determination can be done dynamically, since the chunk size and k are in the Bloom filter RR.

Since the content of any given chunk is static during the lifetime of that zone instance, chunk URLs can be cached or distributed by any content distribution network. To avoid confusion and cache coherency issues at zone change time, we recommend that the SOA serial number for the zone be included in the filename portion of the URL.

7. Dealing with False Positives

As noted, false positives are possible with Bloom Filters. The implications differ for different possible uses of the technology.

For authenticating NXDOMAIN requests, there is no ready recourse. A false positive means that a name server has returned an error report for a domain that the Bloom filter claims exists. This could be a false positive, or it could be the exact form of attack that the Bloom filter mechanism is intended to prevent. Palliative measures include retrying the query from a client not believed to be under attack, or waiting for a new instance of the zone file, and hence a different bit array (see below).

The situation is brighter for opt-in secure zones. In this latter case, the bit array represents zones for which signature records should exist. A server building a bit array can check the remaining names in the zone to see if there are any false positives. If so, a NOKEY record can be generated for such names. This greater tolerance of false positives permits selection of filter parameters that yield smaller bit array sizes and/or fewer hash function calculations. Of course, that must be traded off against the extra signed NOKEY records.

8. Hash Function Families

The behavior of Bloom filters depends strongly on the quality of the hash functions that are used. One good choice is to use SHA2-512 [SHA2], which produces 512 bits of uniformly distributed output. This can be divided into 16 32-bit hash values, which in turn can be reduced modulo m to represent the output of 16 hash functions. If more hash functions are needed, a counter can be concatenated with the the name:

```
SHA2("1" || name)
SHA2("2" || name)
...
```

If we set k to 8, we can use SHA2-256, which is significantly cheaper.

To avoid persistent problems from false positives, it may be desirable to change the hash function for each new zone instance. This is most easily done by including the zone serial number in the hash:

```
SHA2("1" || name || serial)
```

```
SHA2("2" || name || serial)
...
```

Open issue: is this desirable? It leads to more unpredictability in behavior from day to day. On the other hand, addition of any new records to the zone can generate new false positives.

It is not clear that the cryptographic properties of SHA2 are helpful here. There does not appear to be any advantage to an enemy who can deliberately cause collisions. Accordingly, it might make sense to investigate cheaper hash functions.

9. Performance Issues

To process a zone as suggested above, one or two invocations of SHA2-512 per name are needed. Signing an RRset would require a single invocation of a cheaper hash function (probably SHA1 [RFC3174]) per name, plus a very expensive digital signature operation. Until the percentage of signed RRsets becomes quite high, it is clear that Bloom filters are much cheaper.

Chunk size determination relies on the tradeoff between the number of chunk signatures that must be computed versus the bandwidth needed to retrieve chunks. In general, one should opt for more signatures and smaller chunks; the signing operations happen once per zone, while the chunks are retrieved frequently. A 1 KB chunk size is not unreasonable, but that would require 75,000 signatures for a 75 MB bit array.

Bit array size per se is not a major issue, unless many replicas of the data are desired.

10. Dynamic Update

Bloom filters are not compatible with certain forms of dynamic updates. However, the problem caused is bounded and often manageable. Ultimately, the acceptability will depend on the rate of dynamic updates and the number of chunks used.

Deleting records is easy: do nothing. A deleted record will still appear in the bit array, but that manifests itself as a false positive, a problem inherent to Bloom filters. A new filter should be calculated and distributed when the number of deletions has raised the false positive response probability to an unacceptable level. If necessary, the initial selection of k and m can be adjusted to compensate for some predicted rate of deletions.

Additions are trickier. Conceptually, all that is necessary is to calculate the new bit positions that need to be set to 1; however, the existence of signed chunks complicates the matter. The exact behavior will depend on the addition rate and upon the number of chunks.

Without trying to calculate the exact probability distribution, it is clear that the number of chunks changed per unit time is bounded by the product of k and the number of update operations. As long as the computer signing the chunks can keep up with that rate of operations, there should not be a problem. And the network bandwidth required is minimal; all that needs to be sent out is a set of new signatures, plus the bit positions that must be turned on in each chunk. In particular, it is not necessary to redistribute the entire bit array.

11. IANA Considerations

New families of hash functions may be defined and registered with IANA. Registration may be done only by means of a standards-track RFC.

12. Security Considerations

If the signatures specified here are checked, there do not appear to be any correctness issues. However, the chunk retrieval protocol may be abused to flood the server. Note that this server need not be co-located with the zone servers; doing that would limit the effect of such an attack.

As with other aspects of DNSSEC, replay attacks are possible. An enemy could return a stale -- but signed -- Bloom filter RR in response to a query. Similar attacks can be carried out against the chunk retrieval protocol, but not against the TLS variant.

13. References

[Bloom70] "Space/time trade-offs in hash coding with allowable errors". Bloom, B. H. Communications of ACM 13, 7 (July 1970), pp. 422-426.

[RFC2535] "Domain Name System Security Extensions". D. Eastlake. March 1999.

[RFC3174] "US Secure Hash Algorithm 1 (SHA1)". D. Eastlake. September 2001.

[SHA2] "Secure Hash Standard", Draft Federal Information Processing Standards Publication 180-2, May 2001.

14. Author Information

Steven M. Bellovin
AT&T Labs Research
Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07932
Phone: +1 973-360-8656
email: bellovin@acm.org

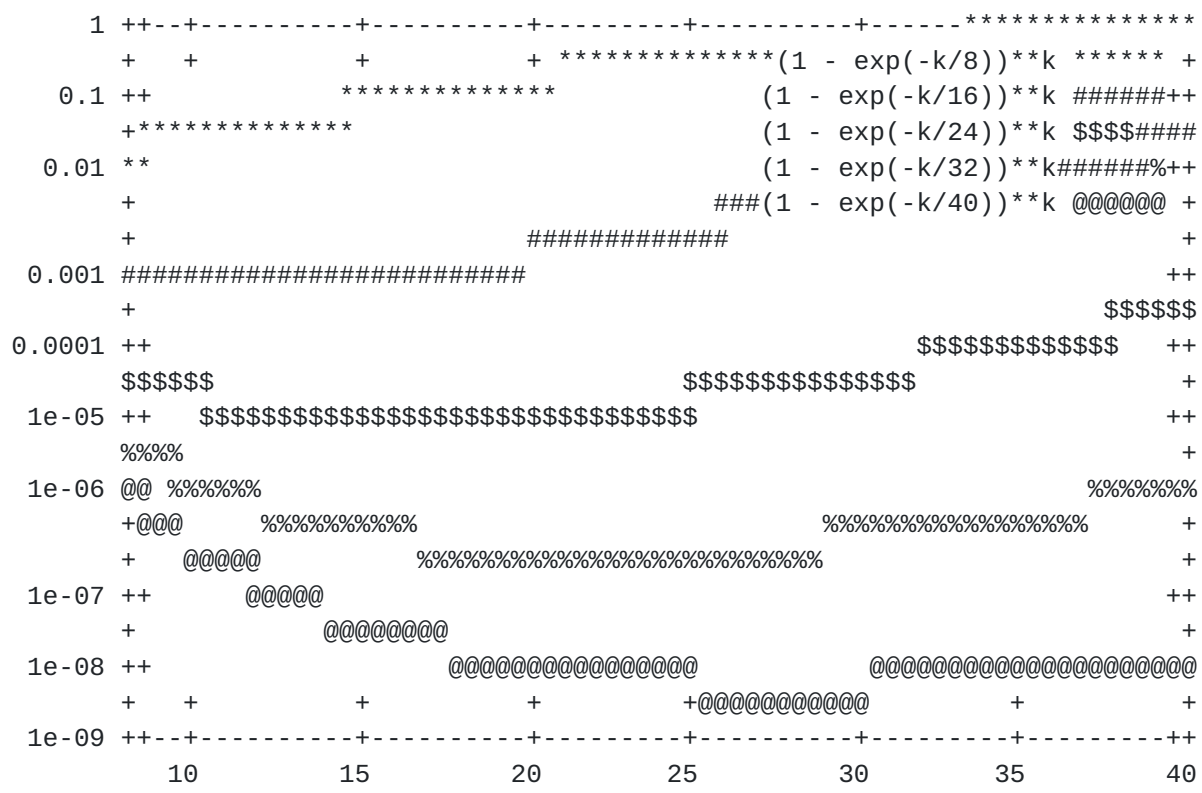
A. Appendix: Bloom Filter Parameters

Below is a table showing the false positive probability p for various values of k and ratios of m/n. We set n -- the number of entries in the zone -- to 25,000,000, the approximate size of .COM at this time.

While the actual choices dependon the acceptable false positive rate, the choices of (8,32), (8,40), (16,32), and (16,40) seem to produce favorable ratios of false positive rate to chunk size and bit array size.

p	k	m/n
0.000000005065	**24**	40
0.000000005112	**32**	40
0.000000010765	**40**	40
0.000000019475	**16**	40
0.000000216758	**24**	32
0.000000330046	**16**	32
0.000000422277	**32**	32
0.000001165717	**8**	40
0.000001366603	**40**	32
0.000005731505	**8**	32
0.000009874368	**16**	24
0.000016565253	**24**	24
0.000041690856	**8**	24
0.000055936473	**32**	24
0.000230939749	**40**	24
0.000574496205	**8**	16
0.000649828308	**16**	16
0.002335383727	**24**	16
0.009530761107	**32**	16
0.025491730341	**8**	8
0.032516117391	**40**	16
0.097625617676	**16**	8
0.293563779663	**24**	8
0.553477428654	**32**	8
0.763051324818	**40**	8

The following (ugly) ASCII graph summarizes the calculations:


1
0.1
0.01
0.001
0.0001
1e-05
1e-06
1e-07
1e-08
1e-09
(1 - exp(-k/8))**k
(1 - exp(-k/16))**k
(1 - exp(-k/24))**k
(1 - exp(-k/32))**k
(1 - exp(-k/40))**k
10
15
20
25
30
35
40