# The ISP Column

*An occasional column on things Internet*

August 2006

*Geoff Huston*

## DNSSEC - The Theory

In looking at the general topic of trust and the Internet, one of the more critical parts of the Internet's infrastructure that appears to be a central anchor point of trust is that of the Domain Name Service, or DNS. The mapping of "named" service points to the protocol-level address is a function that every Internet user relies upon, one way or another.

The ability to corrupt the operation of the DNS is one of the more effective ways of corrupting the integrity of Internet-based applications and services. If an attacker can in some fashion alter the DNS response then a large set of attack vectors are exposed. An altered DNS response may cause some users of your domain name to believe that your services are associated with the addresses of the attacker's servers. For these users, any attempts to communicate with your servers will be redirected by the DNS to communicate with the attacker's servers. Web servers, email, VOIP services, and indeed any service where the initial connection is made through the DNS is vulnerable to such forms of attack on the DNS.

DNS was designed as a distributed database to enable it to scale and to perform extremely efficiently under a wide variety of operational conditions. There are, unfortunately, many ways in which the Internet's name resolution function is vulnerable to attack, and it appears that the DNS has been the subject of many forms of attack over the years.

But the purpose of this article is not intended to be a collection of recipes on how to attack the DNS, or even a description of the weaknesses of DNS. RFC 3833 contains an excellent summary of the vulnerabilities in the DNS. The more useful question is whether it is possible to strengthen the DNS. The DNS is a query - response application, and the critical question in terms of strengthening its function is whether it is possible to authenticate the answers provided by the DNS.

One of the longstanding approaches to this question of authenticity of data is that of public key cryptography.

### A digression into public key cryptography

Communications networks support interaction at a distance. I can't touch you, see you, or hear you, nor can you touch, see or hear me, yet we need to exchange information, or messages. When I receive a message from you, how can I be absolutely sure that it was you who originally sent it? How can I be assured that the message has not been tampered with? How can I be assured that you cannot subsequently deny that you sent this message? And if the message was intended to be a private communication between just the two of us, then how can we ensure that no one else other than myself, the intended recipient, can read this message? None of these objectives are particularly novel, and the

various forms of response to these rather demanding requirements over the years forms the rich and colourful history of cryptography.

The means of assuring communications is through the use of secrets. If you and I can meet in some secure location, and swap a collection of secret "keys" or cryptographic seeds, then I can encipher my messages to you using my private enciphering key and then pass the result into a second encipher process using your shared secret encipher key. If anyone intercepts the message it is supposed to be unintelligible. When you receive my message you first decipher it with your private decipher key, and then decipher the result with my shared decipher key.

Some terminology that may help a bit here: If the key is *symmetric* the encipher and decipher keys may be the same. If the key is *asymmetric* the encipher and decipher keys will differ. The asymmetric keys may be *one way*, in which case the decipher key can only decipher material that was enciphered by the corresponding encipher key, or they may be *two way* in which case either key can be used to encipher material that only the corresponding partner key can decipher.

But this is still not very useful. Its not clear that we are ever able to actually meet and exchange keys, nor is it clear that we are able to meet every time I want to update my keys.

What we would like is an asymmetric two way key pair, such that messages enciphered using one of the keys can only be deciphered using the other key, and knowledge of one key can never lead to deducing the value of the other key in the pair. If one key value is kept as a closely guarded personal secret, and the other is made public knowledge, then we have managed to provide a useful framework for secure messages.

This form of key pair is very useful. If I encipher a message using your public key, then only your private key can decipher it. Similarly if I encipher a message using my private key, then only my public key can decipher it. If I use my private key to encipher a message, and then encipher the result using your public key then I can construct a message that only I could've generated and only you can read. So now we know how useful such a key system can be, do such keys actually exist?

One of the keystones to public key cryptography lies in the area of modular mathematics. Imagine two functions (f() and g()) that are related such that one is the logical inverse of the other. Applying function f and then function g to a value will result in the same original value. Equally the same result happens when applying g and then f to the value I.e. $x = f(g(x)) = g(f(x))$. For example, f() can be "add 1" and g() can be "subtract 1". Now lets add a further constraint, that knowledge of one function provides no clue as to the other function. Simple additive and multiplicative functions fail with this latter constraints, as do exponentiation functions. However, an example of such a pair of functions can be constructed using exponentiation modulo a prime value. The basic mathematical property that is exploited by public key cryptography is given a *prime* p, and a *primitive* g, then $(g^a \bmod p)^b = (g^{ab}) \bmod p = (g^b \bmod p)^a$.

Now we appear to be getting close to a desired outcome. As long as you are confident that the public key that you believe is mine really is mine, and that I have been careful and never divulged my private key to anyone else, then we can set up a secure communication. I can send you messages that are encrypted using my private key, that only I could've generated as only my public key 'unlocks' the message, and I cannot subsequently repudiate that I was the author of the message. If I apply a second encryption pass using your public key over the message once I've used my private key, then only your private key can unlock this operation, resulting in a tamper-proof message that only I could've sent and only you can read.

It would appear that the critical assumption behind this operation is that the public key that you believe is my public key really is my key. How can you be more confident about this assumption? This is perhaps the most significant question in any public key cryptography system. Again if we could meet, and exchange credentials and public keys then the problem is solved for a while, or at least until either you or I choose to re-key. But what if we can never meet? How can we trust the authenticity of our alleged public keys?

One way to help is to use chains of knowledge and role authorities to create chains of trust. If you and I both trust some third party, and this third party is willing to vouch for my public key, then you have some grounds to trust my public key, as long as you trust this intermediary third party.

This chain of trust indirection can be lengthened so that you trust party A, who trusts party B, and so on, and I trust party Z, who trusts party Y, and so on, then if these two chains of trust intersect, then we may have grounds to believe the public key assertions. There are various ways to construct such chains of trust, using strict hierarchies, using arbitrary bilateral trust relationships or using a collective trust threshold. The resultant structure, whatever its chosen form, is generally collectively referred to as a "public key infrastructure".

The DNSSEC question becomes one of establishing whether it is possible to use public key cryptography to confirm that the answer received as a result of a DNS query is indeed the answer that corresponds to the current authoritative state of the DNS.

With that in mind lets have a look at DNSSEC.

## What is DNSSEC?

Well, firstly, lets look at what DNSSEC is not. DNSSEC is not a security panacea. It is not a robust defence against all forms of attack against DNS servers and clients. DNSSEC is not an exercise in encryption of DNS data.

> Within the scope of DNSSEC, DNS protocol interactions remain in the clear, and DNSSEC does not attempt to construct secured private DNS realms. Encryption of DNS exchanges using mechanisms such as TSIG for data protection between DNS servers, are orthogonal to the relatively focussed DNSSEC objective of allowing clients to authenticate the contents of a DNS response.

DNSSEC is a specification of an extension to the DNS through the definition of additional DNS Resource Records that can be used by DNS clients to validate the authenticity of a DNS response, the data integrity of the DNS response, and where the response indicates no such domain or resource type exists, this negative information can also be authenticated. In other words, if an attacker attempts to create a DNS response that has been altered from the original authentic response in some fashion, and the attacker then attempts to pass the response off as an authentic response, then a DNSSEC-aware DNS client should be able to detect the fact that the response has been altered and that the response does not correspond to the authoritative DNS information for that zone. In other words, DNSSEC is intended to protect DNS clients from forged DNS data. This protection does not eliminate the potential to inject false data into a DNS resolution transaction, but it adds additional information to DNS responses to allow a client to check that the response is authentic and complete.

To achieve this, DNSSEC defines a number of new DNS resource records (RRs), namely the **DNSKEY**, **RRSIG**, **NSEC** and **DS** RRs, and two new message header bits: checking Disabled (CD) and Authenticated Data (AD), and it relies on functions provided by Extended DNS mechanisms (EDNSO). With DNSSEC a zone administrator "digitally signs" a Resource Record Set (RRSet), and publishes this digital signature, along with the zone administrator's public key, in the DNS. In checking a DNS response, a DNSSEC client can retrieve the related RRset digital signature and then check this signature using the public key against the locally calculated hash value of the RRset, and then validate the zone administrator's public key against a hierarchical signature path that leads to a point of trust. If all these checks succeed than the client has some confidence that the DNS response was complete and authentic.

DNSSEC implies different actions for different roles. For a DNS zone administrator, DNSSEC is essentially the process of signing RRSets with a private key, publishing these signatures for each RRset in the zone file, and publishing the zone public key in the zone file. In addition the zone administrator has to get the zone’s public key signed by the parent zone administrator. For a DNS client DNSSEC is the ability to perform a number of additional checks on a DNS response that can result in greater trust in the authenticity and accuracy of the DNS response. And for the DNS itself DNSSEC essentially represents a number of additional Resource Records that hold digital signatures of DNS information, as well as key information.

## DNSSEC Resource Records

DNSSEC introduces four additional RRs to carry security information. These resource records are:

### DNSKEY:

Every DNSSEC secured DNS zone has an associated private and public key pair, as generated by the zone's administrator. The private key remains the (closely guarded) secret of the zone administrator. The associated public key for the zone is published in the zone file in the form of a DNSKEY resource record.

An example DNSKEY record for the zone example.com is as follows [from RFC4034]:

```
example.com. 86400 IN DNSKEY 256 3 5 ( AQPSKmynfzW4kyBv015MUG2DeIQ3
                                       Cbl+BBZH4b/0PY1kxkmvHjcZc8no
                                       kfzj31GajIQKY+5CptLr3buXA10h
                                       WqTkF7H6RfoRqXQeogmMHfpftf6z
                                       Mv1LyBUgia7za6ZEzOJBOztyvhjL
                                       742iU/TpPSEDhm2SNKLijfUppn1U
                                       aNvv4w==  )
```

The Time to Live (TTL) value is 1 day (86400 seconds). The Flags value is 256, indicating that this is a Zone Key. The protocol value is the constant value 3. The next field is the public key algorithm, and the value 5 indicates RSA/SHA1. The RR value is the Base64 encoding of the public key value.

**RRSIG:**

A "Resource Record set" (RRset) is a collection of RRs in a DNS ZONE that share a common name, class and type. In DNSSEC RRsets are digitally signed by the zone administrator. This signature is generated by generating a hash of the RRset, then encrypting the hash using the zone administrator's private key. For a zone that contains SOA, NS, A, MX, DNSKEY resource records there are, minimally 5 distinct RRsets, and each RRSET would have its own RRSIG Resource Record. This implies that the granularity of DNSSEC signing is not that of an entire zone, but is aligned to a unit of a DNS query response.

An example RRSIG record for the zone example.com is as follows [from RFC4034]:

```
host.example.com. 86400 IN RRSIG A 5 3 86400 20030322173103 (
                                20030220173103 2642 example.com.
                                oJB1W6WNGv+ldvQ3WDG0MQkg5IEhjRip8WTr
                                PYGv07h108dUKGMeDPKijVCHX3DDKdfb+v6o
                                B9wfuh3DTJXUAfI/M0zmO/zz8bW0Rznl8O3t
                                GNazPwQKkRN20XPXV6nwwfoXmJQbsLNrLfkG
                                J5D6fwFm8nN+6pBzeDQfsS3Ap3o= )
```

The first four fields specify the owner name, TTL, Class, and RR type (RRSIG). The next field is the Type Covered field, and the value of "A" indicates that this is a signing of the A RRs for "host.example.com". The next field is the signing algorithm, and the value of 5 indicates that this is signed using RSA/SHA1. The value 3 is the number of Labels in the original owner name. The value 86400 in the RRSIG RDATA is the original TTL value for the covered A RRset. 20030322173103 and 20030220173103 are the expiration and inception dates, indicating that the RRset signature was created on 17:31:03 20/02/2003, and the signature expires at 17:31:03 22/03/2003. The key tag is 2642 and the signer's name is "example.com." The remainder of the RR value is the encrypted hash of the RRset.

**NSEC:**

The DNSKEY and RRSIG records can be used to check the authenticity of a DNS response, where there is a DNS response, but where there is no authoritative data to return then authentication requires additional information.

The NSEC RR can be considered as a "gap spanning record" for authentication purposes. The entire zone file is ordered in a canonical form, and then NSEC RRs are added to cover the "gap". If the zone contained the names "alpha" and "beta", then there would be a NSEC RR for alpha, and the RR value would be beta, indicating that there are no defined names that lie between "alpha" and

"beta". In addition, the NSEC record defines the set of RR types for this domain name. Continuing the example, the NSEC record for "alpha", would have as a value field the enumeration of the RR types that are defined for "alpha".

In response to a query, if the name does not exist in the zone file, or the RR type does not exist for the name in question, then the NSEC record is returned as authenticatable evidence that the name, or the RR type does not exist.

The reason for this form of spanning data, as distinct from a more generic response of "no such name", is that the latter form of response can be used in a replay attack, falsely claming that a name does not exist, whereas the NSEC record explicitly informs the client of the "gap".

An example NSEC record for the zone example.com is as follows [from RFC4034]:

```
alfa.example.com. 86400 IN NSEC host.example.com. (
                                A MX RRSIG NSEC TYPE1234 )
```

The first four text fields specify the name, TTL, Class, and RR type (NSEC). The entry host.example.com. is the next authoritative name after alfa.example.com. in canonical order. The A, MX, RRSIG, NSEC, and TYPE1234 mnemonics indicate that there are A, MX, RRSIG, NSEC, and TYPE1234 RRsets associated with the name alfa.example.com.

NSEC records can be used to enumerate the entire contents of a zone file. While DNS data is public information, there is some concern over the implicit publication of the entire zone in this manner. We'll come back to this later with the NSEC3 RR.

**DS:**

> The perennial issue in public key cryptography is that of the means of validation of the public key. If an attacker manages to intercept all your traffic, signs responses using their own private key and represents their public key as that of the party you thought you were communicating with, then how are you to tell that the communication has been corrupted? In public key cryptography the conventional answer is to find other parties who are willing to attest as to the validity of a public key, and find yet more parties who are willing to attest as to the validity of the first group of attesters, and so on until you find a link with a party that you trust. Assuming that trust is a transitive quantity (always a risky assumption of course), then you can construct a chain of trust from your trust point to the party whose public key you are attempting to validate.

The issue of validation of the zone public key remains unaddressed with the first three Resource Record types. An attacker would simply need to supply the DNSKEY and RRSIG data to match the bogus RRset data in order to make the response look 'authentic'. So we are back to the same public key validation question - how can a client validate the DNSKEY record?

The approach adopted by DNSSEC is to use a chain of trust within the hierarchical delegation structure of the DNS itself. Apart from the root zone, every DNS zone has a parent zone. The Delegation Signer (DS) RR contains the hash of the public key of the child zone. This record is

signed by the parent zone's private key with a matching RRSIG RR. To validate a zone's DNSKEY, the associated DS, RRSIG(DS) and DNSKEY of the parent zone is retrieved. The DS record is validated by using the DNSKEY to encrypt the RRSIG(DS) record, and then checking that the result matches the DS record. This is the zone public key, according to the zone's parent. This can be compared to the DNSKEY record of the zone in question. This relies on the parent zone key, and the question is how can this key be validated? The same process can be applied here. The process stops when the DNSSEC client encounters a "trusted" key. The ideal "trust key" would be the DNSKEY of the root zone, but in the absence of such a trust anchor each DNSSEC client has to configure their trust validation system with known trust points where there is no parent validation.

An example DS record for the zone example.com is as follows [from RFC4034]:

```
dskey.example.com. 86400 IN DS 60485 5 1 ( 2BB183AF5F22588179A53B0A
                                           98631FAD1A292118 )
```

The first four text fields specify the name, TTL, Class, and RR type (DS). Value 60485 is the key tag for the corresponding "dskey.example.com." DNSKEY RR, and value 5 denotes the algorithm used by this "dskey.example.com." DNSKEY RR. The value 1 is the algorithm used to construct the digest, and the rest of the RDATA text is the digest in hexadecimal.

## DNSSEC Properties

There are some very useful properties of this DNSSEC approach to the use of public key cryptography to the DNS.

Signing is at the granularity of a DNS response, while keys are at the level of granularity of a domain zone. While an entire DNS zone has a single key pair, the entire zone file is not signed - individual RRsets are signed. When thinking of DNS as a transaction protocol this makes a lot of sense, as the major use of DNS is in posing queries that are answered by RRset responses. A signed zone would imply that authentication of a single response would entail an entire zone transfer, which is a preposterous overhead for a simple DNS transaction! The RRset signing model also implies that incremental changes to a DNS zone file entail generating new signature objects (RRSIG RRs) only for those RRsets that have been altered. Again this minimizes the amount of overhead in key generation.

“No data” responses can be signed. In DNS the "no such domain" and “no such type” responses to queries are as important as returned information. Of course its not possible to add authentication information to such non-responses, and DNSSEC addresses this problem by generating "gap" NSEC records as a means of authenticating the DNS gaps. This concept of using synthetic information as material to assist in authentication of a non-response is an interesting approach.

Validation exploits the DNS delegation hierarchy. Validation of the public keys in a zone exploits the DNS delegation model and uses the parent as the means of validating a child zone. It should be noted that its not the identity nor the bona fides of the zone administrator that is at issue here. The implication is that standard identification public key certificates are of little value, as the task is not to validate that the public key “belongs” to a particular individual. The task here is to ascertain if a particular public key is associated with a given zone file. The manner of confirming that association is to rely on the hierarchical structure of delegation within the DNS, and to use the DNS delegation parent to confirm the public key value of the child zone. In theory, with a signed DNS root, one public key, that of the root would need to be distributed out-of-band from the DNS, and all other DNS keys would validate against this single trust anchor. In practice, while a signed root does not exist, and while deployment of DNSSEC is not contiguous, the loading up of trust points for each DNSSEC-equipped remains a significant operational issue.

## How does DNSSEC work?

When requested by the client, DNSSEC adds additional data to the DNS protocol responses that provide additional information to allow the DNS client to authenticate the RRset data response. From the perspective of the protocol transaction between a query agent and an authoritative nameserver, the change is the addition of a RRSIG part to the data response where there is a response that can be generated, and the use of an NSEC RR response, plus its accompanying RRSIG record if there is no authoritative data to respond to the query. In addition to an RRSIG response covering the RRset records in the answer section of the DNS response there is also an RRSIG response covering the records in the authority section and one or more RRSIG responses relating to records in the additional response section.

The client can take the RRset response and use the algorithm referenced in the RRSIG record to generate the hash of the data. The RRSIG value can be encrypted using the DNSKEY public key which will, in effect, decrypt the hash in the RRSIG record. To do this the client must also have at hand the DNSKEY record for the zone. This operation allows the client to check that the hash of the RRset data matches the decrypted RRSIG hash.

The DNSKEY would normally be provided as part of the additional section of a DNSSEC response. If the client has not validated the DNSKEY within some locally defined period, then the client should also validate the DNSKEY value. This entails verifying the RRSIG record on the DNSKEY value, using the same procedure as for the RRset validation. However domain zone key validation also entails the construction of a trust chain back to a trust anchor point. If this domain key is not already a trust anchor then the client needs to query the parent zone for the DS record of the child zone, which returns both a public key value and an RRSIG value, and a DNSKEY RR. This public key needs to be validated using the DNSKEY of the parent zone. This parent zone public key, in turn, must be validated. This iterative process constructs a trust chain that, hopefully, leads back to a trust anchor. At that point the DNS response can be considered to be validated.

## DNSSEC Issues

DNSSEC is not without some additional issues that change the nature of the DNS and have some significant implications for it's performance.

- The average size of a DNS response message increases, due to the additional signature records that are attached to the response. Where the message size exceeds the UDP message size it will need to set the truncated response flag, causing the query to fall back to the use of TCP, with its attendant higher overheads in terms of client and server state, and the number of network messages to manage the TCP connection.

- The number of DNS transactions increases due to the requirement to perform additional queries for zone public key records when constructing trust chains. Even though caching has some potential to reduce much of this traffic, there is still an additional query load that must be considered with DNSSEC.

- The zone file increases in size due to the addition of the additional DNSEC records. The major contributors here are the NSEC and RRSIG records, and the zone size will increase. By what factor depends on what is in the zone file of course, but increases by a factor of up to seven in size have been noted in DNSSEC literature.

- The client has to spend additional time validating the signed data and validating the public key, potentially slowing the resolution process.

- The server has to generate new signatures over all RRset changes, which places an incremental load on the server function.

- As noted in RFC3833 DNSSEC is complex to implement, and testbed experience to date suggests that trivial zone configuration errors or expired keys can cause serious problems for a DNSSEC-aware resolver.

- RFC3833 also notes that the behaviour of a small query generating a large response has been used as denial-of-service amplifiers. DNS with DNSSEC makes the DNS an ideal vehicle for this form of denial-of-service attack.

- Key rollover is challenging in most cases, but particularly so in the keys for the DNS root zone. As RFC3833 notes: "Work to date has not even come close to adequately specifying how the root key rolls over, or even how it's configured in the first place."

- the absolute times in the RRSIG records imposes a new regime of some level of time synchronization across the DNS as part of the signature validation process. Previously, DNS had only a concept of elapsed time, whereas now if a validating party has a skewed concept of time it may believe that expired signatures are still valid. If a zone administrator has a skewed time then the signature validity timestamps it generates in its RRSIG records would be incorrect.

Perhaps one of the more significant issues lies in the use of the NSEC response. With judicious use of the NSEC response it is possible to reconstruct the contents of a domain zone file, analogous to the outcome of a DNS list operation. With the significant business interests in the DNS today this implicit zone listing capability is regarded by some zone operators as an exposure of commercially sensitive information that would normally remain private. One view of this exposure, as espoused by Albitz and Liu in the "DNS and Bind" 4th Edition, is the mantra: "My zone data is secure, but public." The other issue exposed with the NSEC response is the behaviour of "split-domain" DNS servers, which will respond with authoritative data to some (presumably trusted) parties, while providing a "no such domain" form of denial response to all others. The vulnerability here is that if the server synthesises a NSEC response to an untrusted party, an attacker could subsequently replay this response to a trusted party, generating a form of denial of service attack. These pre-signed NSEC records expose a considerable amount of information about the zone contents, and this, in turn, has been cited as one of the significant impediments to DNSSEC deployment.

The current response to this issue is the development of an alternate response to the "no data" condition that would still allow the zone administrator to sign the response, but would not necessarily reveal the enumeration of the entire zone as a side effect. This approach, currently work in progress within the IETF, proposes a different RR response, namely the NSEC3 record.

Rather than using the name order within a zone file and explicitly enumerating the next name in the NSEC record, the NSEC3 approach uses a hash algorithm on the names within a zone, and then uses a hashed ordering of these names. The next name references in the NSEC3 RR is the next name corresponding to this hashed name order. The objective of this approach is to increase the cost of zone enumeration using NSEC3 responses.

## DNSSEC Deployment

Deployment of DNSSEC has been quite piecemeal at present, and it has yet to gather any significant momentum so far. There are a number of reasons why this is the case.

The first, and perhaps the hardest, is that the Internet of today is now a very large system, and every large system tends to resist change. Any change, and particularly a change to a technology that is as universally pervasive as the DNS, will take time, and the larger the realm of deployment, the longer the period to get to a critical mass of deployment.

There are a few other issues with DNSSEC that hinder deployment. One is that the economics of DNS deployment do not work directly in favour of DNSSEC. It would be good if a security measure could create outcomes that were simpler, cheaper and more robust, as well as creating mechanisms to deflect various forms of hostile attack. DNSSEC would not readily be described in such terms. There are additional duties placed on the zone administrator, additional load placed on DNS servers, additional responsibilities placed on DNS clients, and the potential for additional tasks to be undertaken by applications. Even in such a situation, deployment proceeds where the cost of the measure is offset by direct benefits achieved through the measure. Here it is not clear that there is a cost and benefit equation at work for each player. The DNS zone administrator inherits a significantly more complex issue, for the existing issues of securing the DNS servers and ensuring that they can resist various forms of attack do not go away with DNSSEC, and on top of that DNS places an additional workload of key management and signature generation for each change to the zone. The benefits that arise from this additional cost are not readily apparent to the DNS zone administrator or to the DNS server system administrator. The initial benefit would appear to accrue to the client, whose additional work in validating the DNSSEC responses would lead to a greater level of confidence in the accuracy in the DNS. But perhaps the real benefits accrue to third parties who are then able to view the DNS as a relatively efficient means of distribution of authenticable data. The work in looking at a CERT RR to store a certificate in the DNS as a resource record, and then use this certificate as a means of injection of authentication into applications, such as email could be of significant benefit to such applications. But when cost and benefit are decoupled to such an extent there is often significant impediments to deployment.

The other part of the consideration of the economics of DNSSEC is the observation that all security measures are ultimately an exercise in risk evaluation, and the cost of deployment of the measure has to be evaluated against the practical probability of attack and the potential consequent costs of the attack. DNS cache poisoning is not a prevalent form of attack in current DNS implementations, and DNS resolvers are relatively adept at resisting various forms of cache poisoning. That does not imply that the DNS is perfect in this respect, but it does make the business case for deployment of DNSSEC one that requires some considerable thought.

The other aspect of DNSSEC that appears to be a hindrance rather than an asset is that piecemeal deployment of DNSSEC makes DNSSEC resolution harder in some respects rather than easier. The issue here is that of the identification of trust anchors in the key chains. DNSSEC does not permit indirection of any form, and either the zone's immediate parent must be in a position to countersign the zone's public key, or the zone must become a trust anchor in its own right. How does a DNS client establish a trusted relationship with all the current DNSSEC signed zones which have no immediate DNSSEC delegation parent? How does a DNS client become aware of zone key rollovers of these trust points? How can this process be managed in some automated fashion outside of the DNS delegation hierarchy?

Of course the bifurcation of the "no data" response with NSEC and NSEC3 RRs does not help DNSSEC either. Irrespective of the relative level of deployment of zones using NSEC and NSEC3, a DNS client must be equipped to handle both forms of response, making the client more complex as a result.

And, of course, there is the ultimate issue of use of DNSSEC. It has been commonly reported that when a browser generates a pop-up screen warning that a browser certificate has expired and requesting whether the user wishes to proceed in any case, the common response is to proceed! What form of application behaviour would be appropriate in the case of a DNSSEC validation failure. In applications that are intended for direct interaction with a human end user, then there is always the option of a dialog box and some user direction as to how to proceed. But DNS resolution occurs in

many contexts, and it is often the case that there is no option to enter into a user dialog as to how to proceed. Should an application deliberately fail if DNSSEC validation fails? Would we be opening up a new form of denial of service attack by deliberately corrupting a DNSSEC response in order to trigger application failure on the end system?

On the other hand, there is the view that exposure of vulnerabilities leads inexorably to exploitation, and leaving this form of DNS-based attack open will lead to more prevalent forms of exploitation. From this perspective measures to allow a client to validate that a DNS response is authentic and complete is a valuable tool, and deployment experience should stimulate the creation of DNS zone management tools that master the inherent complexities of this technology.

## Conclusion

DNSSEC is a mechanism that allows a client to validate DNS responses. It can expose attempts to pass off false DNS data as authentic. DNSSEC is not an instance of a public Key Infrastructure, in that there are no public key certificates, no revocation capability, and no explicit identification of the parties involved.

However the introduction of signed data into the DNS does have some potential for other uses of the DNS that are related to key distribution mechanisms. The use of public key certificates in the DNS in the context of key distribution for SSH or IPSEC are potential candidates here. Another interesting direction is the activity of Domain Keys Identified Mail (DKIM), where it is feasible to use such certificates in the DNS as a means of providing additional information to assist receiving domains in detecting certain forms of spoofing within email.

So that's the theory part of DNSSEC. The practice part, of actually creating a DNSSEC zone, is a task that I'll report on in a future column, once I've managed to set it up correctly!

## DNSSEC reading


RFC 3833 A Threat Analysis of the Domain Name System
RFC 4033 DNS Security Introduction and Requirements
RFC 4034 Resource Records for the DNS Security Extensions
RFC 4035 Protocol Modifications for the DNS Security Extensions
RFC 4398 Storing Certificates in the Domain Name System (DNS)
NSEC3 – "DNSSEC Hashed Authenticated Denial of Existence" draft-ietf-dnsext-nsec3-06.txt
DNS and Bind, 4th Edition, Paul Albitz and Cricket Liu, O'Reilly.
www.dnssec.org – A resource page for DNSSEC

## Disclaimer



## About the Author

*Geoff Huston* B.Sc., M.Sc., has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. He is author of a number of Internet-related books, and has been active in the Internet Engineering Task Force for many years.

*www.potaroo.net*