June 2010

Geoff Huston

DNSSEC – A Review

After many years, the root of the DNS is evidently going to be signed in the coming weeks using DNSSEC with a verifiable root key, or at least that's the plan if the National Telecommunications and Information Administration of the United States Federal Department of Commerce follow through with their proposed actions that have been foreshadowed in the Federal Register of the United States bureaucracy of Wednesday, 9 June 2010. It will all happen by July 15 2010, if all happens in accordance with the plans outlined in that notice, and on that date we should have a DNSSEC-signed root of the DNS. Given that this is an event that has taken more than fifteen years to come to fruition, I thought it might to useful to have another look at DNSSEC to mark this long anticipated milestone.

Figure 1 The View of DNSSEC in 2006 – T-Shirt by Olaf Kolkman

DNSSEC and the DNS

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. While almost every single user transaction deals with names, such as `www.potaroo.net`, every single network packet has to deal with protocol level addresses, such as `203.119.0.116` or `2401:2000:6660::2`.

Given that every single Internet user and every Internet service provider ultimately relies on the integrity of the DNS, then how is the DNS itself protected against malicious attack? If the DNS is the cornerstone upon which the integrity of operation of all other user level Internet applications are constructed, then how can we be assured that the DNS itself is working as intended?

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 a malicious attacker can, in some fashion, alter a DNS response without being detected, then a large set of attack vectors are exposed.

For example, an altered DNS response may cause some users of your service to believe that your online services are associated with the addresses of the attacker's servers. For those unfortunate users who attempt to communicate with your servers, their traffic will be redirected by the DNS to communicate with the attacker's servers. While at one level this can appear to be simply an annoying piece of vandalism, when the same form of attack is directed against financial services or critical public emergency services or other forms of valuable or critical service elements, then this is an extremely serious form of attack with serious financial or even political implications. Web servers, email, VOIP services, and indeed any service where the initial rendezvous is made through the DNS is vulnerable to such forms of attack on the DNS, and most services do indeed rely on the DNS.

The DNS was designed as a widely 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. However, 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 [RFC3833].

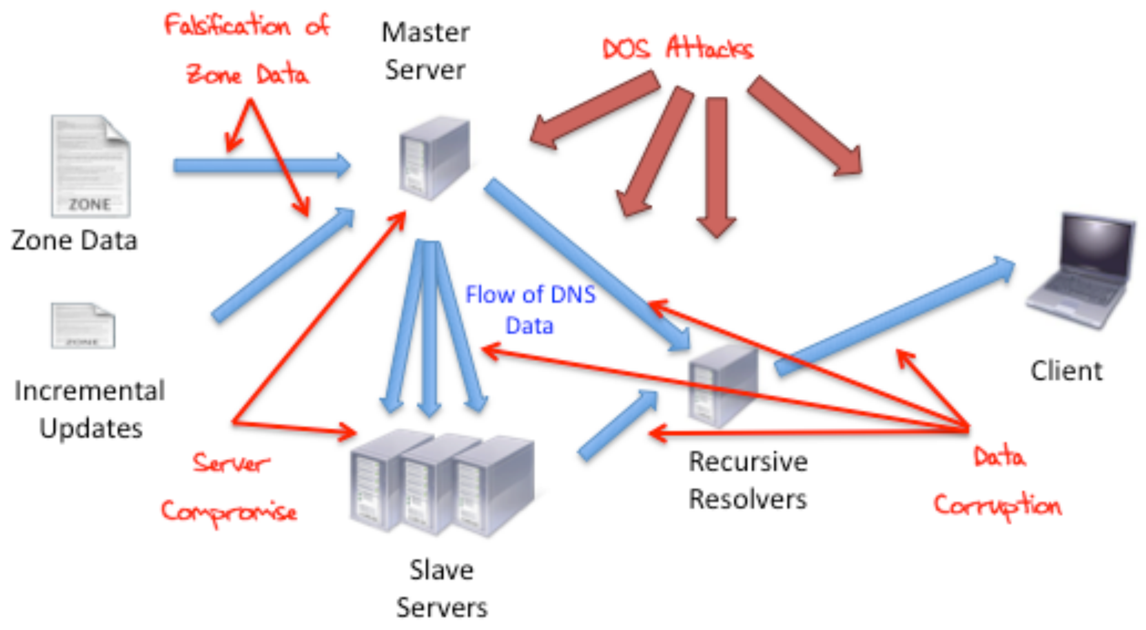


Figure 2 Potential Vulnerabilities in the DNS

The more useful question here is whether it is possible to strengthen the DNS such that such efforts to corrupt the DNS are detectable. The DNS is a query-response application, and the critical question in terms of strengthening its operation is whether it is possible to authenticate the responses provided by the DNS. The question of "securing" the DNS becomes one of establishing whether it is possible to add some form of credentials to DNS responses that would allow the client to check that the data provided in the response corresponds exactly to the data as it was entered into the authoritative zone. The client would like to validate that they have received the entirety of the answer, and that no part has been altered in any way.

The way DNSSEC achieves this is through the use of public key cryptography and digital signing of DNS data. When data is entered into the DNS via the original zone data, the zone data is signed by the zone administrator. When the client performs a DNS query, the signature is provided to the client as part of the DNS response, and the client can verify the signature and in so doing be assured that the response is an accurate representation of the original zone data.

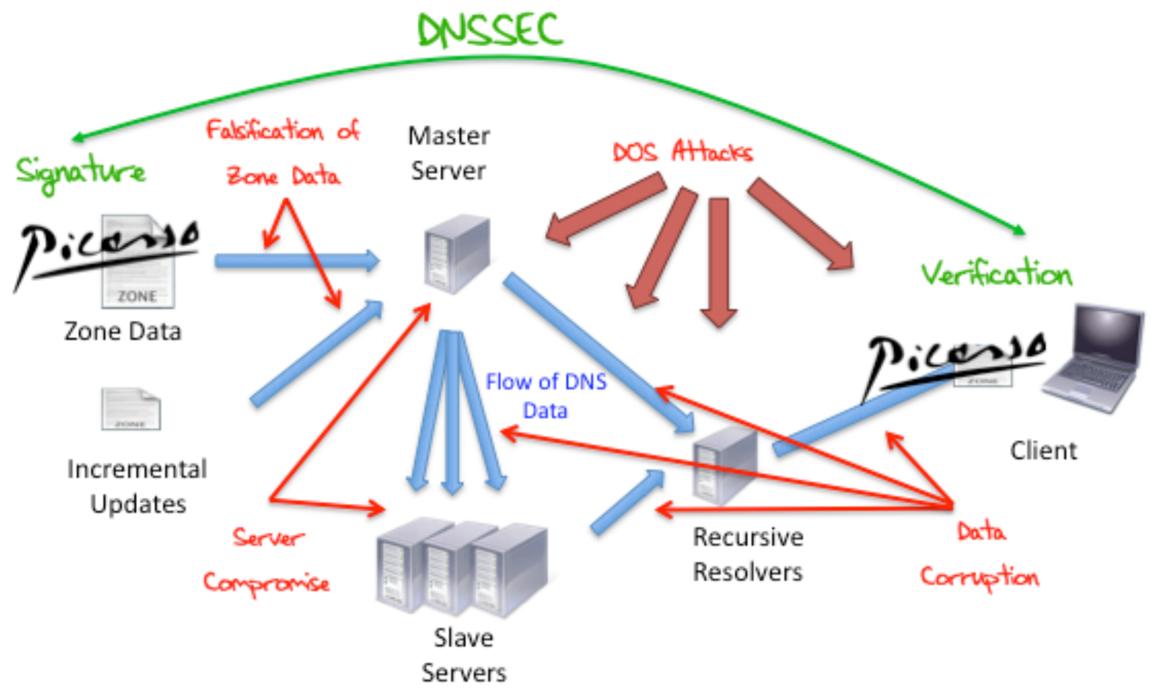


Figure 3 DNSSEC and the operation of the DNS

What is DNSSEC?

In answering this question, it may first help to 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 DNS transactions can be prone to eavesdropping with DNSSEC. DNSSEC does not attempt to construct secured private DNS realms that are protected from third party inspection. Encryption of DNS exchanges using mechanisms such as TSIG for data protection between DNS servers (see RFC 2845), address a different set of functional objectives than that of DNSSEC. DNSSEC does not support secure communications channels between clients and resolvers, nor between primary and secondary servers of a domain. DNSSEC does not prevent various forms of DNS denial of service attacks, nor does it prevent the DNS being used as part of an amplification attack.

The DNSSEC objective is a tightly focused objective of allowing clients to authenticate the contents of a DNS response. It is focused on allowing a client to detect various forms of "man-in-the-middle" attacks where the attacker has modified the original information that is in the authoritative zone file. It does not prevent the possibility of such an attempt to manipulate a DNS response, but it is intended to allow a client to detect that such an alteration of a DNS response has occurred. Again this is not a universal level of protect, but one that is qualified such that the detection is possible in those cases where evidence that the DNS zone from which the DNS response has been assembled has been DNSSEC-signed is still visible to the client.

DNSSEC is a specification of an extension to the DNS through the definition of additional DNS Resource Records (RRs) that can be used by DNS clients to validate the authenticity of a DNS response, the data integrity of the DNS response. In addition, where the DNS response indicates no such name, or no such resource type exists, DNSSEC allows for this negative information to also be authenticated by the client. In other words, if an attacker attempts to create a DNS response that substantively differs 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 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.

A succinct summary of the problem that DNSSEC is intended to address is that DNSSEC is intended to protect DNS clients from believing forged DNS data.

The way this is achieved in DNSSEC is by using public key cryptography (see Appendix I). The DNS zone administrator digitally signs every Resource Record set (RRset) in the zone, and publishing this collection of digital signatures, along with the zone administrator's public key, in the DNS itself. DNSSEC does not make use of Digital certificates or any other form of external credentials. The intention with DNSSEC is to publish all the necessary security credentials in the DNS itself, and use the DNS for both the storage of such security credentials and the means of distribution of this information.

In checking a DNS response for a given RRset, a DNSSEC-aware client can retrieve the related digital signature RRs and then check this signature using the zone administrator's published public key against the locally calculated hash value of the RRset. The client can then validate the zone administrator's public key using 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.

To achieve its objective, DNSSEC defines a number of new DNS RRs, namely the DNSKEY, RRSIG, NSEC and DS RRs (see Appendix II). DNSSEC also introduces two new message header bits: checking Disabled (CD) and Authenticated Data (AD), and it uses functions provided by Extended DNS mechanisms (EDNS). This creates a backwards compatible extension to DNS.

Implementing DNSSEC implies different actions for different roles. For a DNS zone administrator, DNSSEC is essentially the process of signing all RRsets with a private key, publishing these signatures for each RRset in the zone file, and also publishing the public key in the zone file. In addition the zone administrator has to ensure that the zone’s public key is signed by the parent zone administrator and published in the parent's zone. For a parent DNS zone administrator DNSSEC also includes the task of validation of the DNSSEC zone's public keys of the zone's children, signing over them using the zone's private key, and publishing this material as RRs in this DNS zone. 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.

The technical specifications for DNSSEC are in RFC4033 (Security Requirements), RFC4034 (DNSSEC Resource Record Types) and RFC4035 (DNS Protocol Modifications). Further specifications related to the NSEC RR are in RFC4470 (Epsilon Functions for NSEC responses) and RFC5155 (NSEC3 Resource Record). RFC4641 describes DNSSEC operational practices.

How does DNSSEC work?

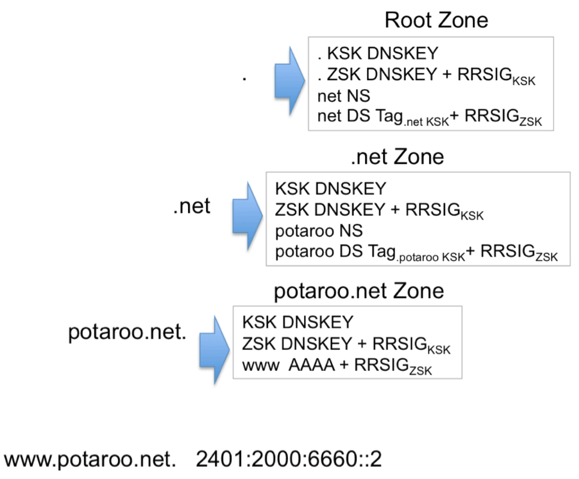
When requested by the client in the DNS query, the authoritative DNS server will add additional DNSSEC data to the DNS responses. This additional data is, in effect the digital signature of the DNS data contained in the response. This is intended to allow the DNS client to authenticate the DNS response. The way in which this is done is by the addition of a RRSIG part to the additional data of the DNS response. If there is no authoritative DNS data to respond to the query, such as when no such domain name exists, then the DNS response will include an NSEC RR response, plus its accompanying RRSIG record. 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.

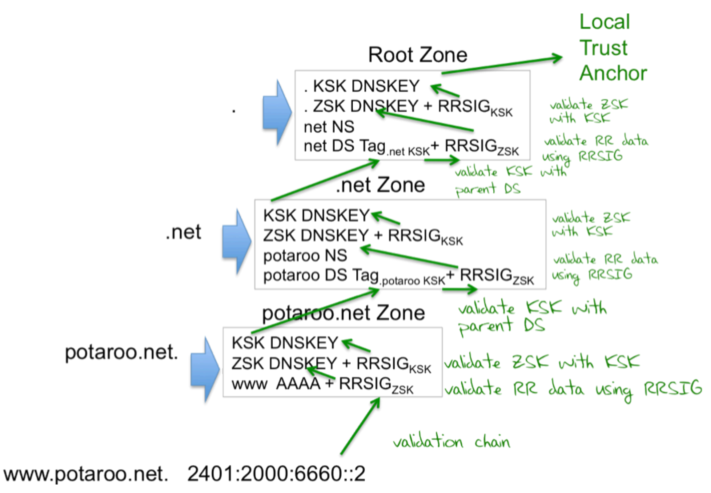
The first task that the DNSSEC-aware client should perform is to use the RRSIG data to check the validity of the DNS response. To do this the client can take the RRset response and use the algorithm referenced in the RRSIG record to generate the hash of the data. The second task is to take the RRSIG value and encrypt it using the DNSKEY public key. To do this the client must also have at hand the DNSKEY record for the zone. This will result in decrypting the hash in the RRSIG record. The results of these two operations are now compared. If the DNS response is authentic then the hash of the RRset data will match the decrypted RRSIG hash value.

The DNSKEY RR would normally be provided as part of the additional section of a DNSSEC response. If the client has not already validated the DNSKEY within some locally defined period, then the client should also validate the DNSKEY value.

The DNSKEY may match a local trust anchor, in which the DNSKEY can be accepted without further tests. Otherwise the DNSKEY RR will need to be validated.

This validation procedure entails verifying the RRSIG record associated with the DNSKEY RR, using the same procedure as described above for other RRs. 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. This DS query should return both a public key value as the value of the DS RR, an RRSIG RR associated with the DS RR, and a DNSKEY RR for the parent zone. The DS RR can be validated against the RRSIG RR, using the public key contained in the DNSKEY RR. This public key, in turn, must be validated. This iterative process constructs a trust chain that, hopefully, leads back to a locally configured trust anchor. At that point the DNS response can be considered to be validated.


Root Zone
.
. KSK DNSKEY
. ZSK DNSKEY + RRSIG$_{KSK}$
net NS
net DS Tag$_{.net\ KSK}$ + RRSIG$_{ZSK}$
.net Zone
.net
KSK DNSKEY
ZSK DNSKEY + RRSIG$_{KSK}$
potaroo NS
potaroo DS Tag$_{.potaroo\ KSK}$ + RRSIG$_{ZSK}$
potaroo.net Zone
potaroo.net.
KSK DNSKEY
ZSK DNSKEY + RRSIG$_{KSK}$
www AAAA + RRSIG$_{ZSK}$
www.potaroo.net. 2401:2000:6660::2


Local Trust Anchor
Root Zone
.
. KSK DNSKEY
. ZSK DNSKEY + RRSIG$_{KSK}$
net NS
net DS Tag$_{.net\ KSK}$ + RRSIG$_{ZSK}$
validate ZSK with KSK
validate RR data using RRSIG
validate KSK with parent DS
.net Zone
.net
KSK DNSKEY
ZSK DNSKEY + RRSIG$_{KSK}$
potaroo NS
potaroo DS Tag$_{.potaroo\ KSK}$ + RRSIG$_{ZSK}$
validate ZSK with KSK
validate RR data using RRSIG
validate KSK with parent DS
potaroo.net Zone
potaroo.net.
KSK DNSKEY
ZSK DNSKEY + RRSIG$_{KSK}$
www AAAA + RRSIG$_{ZSK}$
validate ZSK with KSK
validate RR data using RRSIG
validation chain
www.potaroo.net. 2401:2000:6660::2

DNSSEC: For and Against

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

Record Level Granularity

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 presenting queries that are answered by DNS 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 in response to a single change to the zone file.

Signed "No Data" Responses

In DNS the "no such domain" and "no such resource record type" responses to queries are as important as returned information. Of course it is not possible to add authentication information to such non-responses in the form of a detached signature, and DNSSEC addresses this problem by generating "gap" NSEC records as a means of authenticating these DNS "gaps". This concept of using synthetic information as material to assist in authentication of a non-response is an interesting approach which is not commonly seen in commonly used secure applications.

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 it is not the identity nor the bona fides of the zone administrator that is at issue here. The implication is that conventional X.509 public key certificates are not of direct relevance to this application, as the task is not to validate that the public key "belongs" to a particular individual, or a particular role, nor is the private key holder being granted a particular authority. The objective of the credential structure in DNSSEC is to ascertain that a particular public key is associated with a given zone.

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 an environment of ubiquitous use of DNSSEC, with a signed DNS root, DNSSEC clients would need to maintain just one public key value, that of the DNS root zone. All other DNSSEC zone keys can be verified by a process of backward chaining up to the root zone, following the sequence of DNSKEY and DS records up to the root zone key.

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

Larger Zone Files

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.

Larger Responses

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 maximum 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 behaviour of a small UDP query generating a large response has been used as a denial-of-service amplifier by using a spoofed source address in the DNS query. DNS with DNSSEC makes the DNS an ideal vehicle for this form of denial-of-service attack.

More Queries

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.

Higher Client Overhead

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

Higher Server Overhead

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

Increased Potential for Zone Administration Errors

DNSSEC is complex to implement, and trivial zone configuration errors or expired keys can cause serious problems for a DNSSEC-aware resolver. Key rollover of the KSK presents some challenges, particularly when the parent zone is not DNSSEC-signed.

The use of 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 in DNSSEC if a client 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.

Zone Enumeration with NSEC RRs

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, and the observation that there is more revenue earned in the name registration business through registration of "unused" names than earned from "used" names, 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 is to factor in the observation that with NSEC signature coverage of a zone the zone data is then 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 to use a hash function in the ordering of records 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 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 cannot readily be described in such terms. There are additional tasks 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 in zone administration 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. When cost and benefit are decoupled to such an extent there is often significant impediment to widespread deployment as a consequence.

It is also possible that an indirect benefit of DNSSEC is that the DNS is able to be used as a relatively efficient means of distribution of authenticable data. There is a well established and high value certificate-based reputation industry. One potential use of DNSSEC is to provide an alternate distribution and verification framework for short data items, as a potentially lower cost alternative to certificate-based frameworks. However the history of the DNS is full of inventive proposals to extend the DNS in various ways, and the number of successful and widely adopted extensions is far lower than the number of proposals, so the likelihood of DNSSEC assuming a broader role in service security and integrity and superseding much of the role currently undertaken by certificate-based frameworks is at best a remote possibility.

The other part of the consideration of the economics of DNSSEC deployment 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 without DNSSEC is perfect in this respect, indeed far from it, but it does make the business case for deployment of DNSSEC one that requires some considerable thought. However it should also be remembered that the costs of a successful attack in the DNS are potentially quite considerable.

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 signing key, or the zone must become a trust anchor in its own right. Now if everyone adopted DNSSEC the task of the client would be very straight forward: load the key signing key of the root zone of the DNS into DNSSEC and everything else will validate indirectly via a signing chain against that single trust key. But when there is only piecemeal deployment then a client needs to track down all the instances of local trust points where there is no "upward" signature link back towards the root zone. This poses some quite difficult practical questions: 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? What is the potential role of intermediaries in this process? Who can a client tell between a genuine intermediary and a malign intermediary?

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 for the user to simply request the application 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 widespread use of DNSSEC 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 the DNS without the form of data protection provided by DNSSEC will inevitably 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 with DNSSEC should stimulate the creation of DNS zone management tools that master the inherent complexities of this technology.

Where are we now?

DNSSEC is a standards-based mechanism that allows a client to validate DNS responses. It can expose attempts to pass off false DNS data as authentic using an approach of incrementally signed data and an interlocking hierarchy of signing keys to allow data verification. DNSSEC is a backwards compatible extension of DNS, and it works entirely within the existing structure of the DNS.

However, DNSSEC will work best when we all use it. In such a scenario of universal adoption clients will need to be equipped with a simple single local trust key that will "unlock" the entire key hierarchy of the DNS. If everyone uses DNSSEC we will not need to rely on the integrity and good intentions of a bevy of intermediate crypto key brokers to assemble useful and current lookaside lists to compensate for the lack of a complete interlocking structure of DNS zone keys.

And here "everyone" really means "everyone, from the root all the way down". Within a few days from now the root of the DNS will be DNSSEC signed with a verifiable key. A critical part of the scenario of universal adoption of DNSSEC, that of the apex of the key hierarchy will have been achieved. But when and how will the other parts of the DNS follow? Some DNS registrars are currently offering DNSSEC DS key registration as an extra cost option for name holders. While this provides some marginal incentive from the registration side for DNSSEC adoption, its difficult to see the path towards universal use of DNSSEC taking off when many of the incremental costs are loaded onto the name holder and the zone administrator while the majority of the benefits of adoption of DNSSEC lie in the intangible area of risk mitigation on the DNS client side.

So the signing of the root of the DNS does not in and of itself cure the DNS of all known security pitfalls. This particular root zone signing lifts the expectation that widespread adoption of DNSSEC is a feasible ambition, but its not the complete solution by any means.

It is perhaps more realistic to view the signing of the root of the DNS as one more milestone, albeit a very important and significant milestone, in a far longer DNSSEC path, rather than thinking that we've reached the ultimate destination of this particular journey.

References and Further Reading

The Code Book by Simon Singh, Anchor, ISBN: 978-0385495323, August 2000.
A general reading background in cryptography.

Network Security: Private Communication in a Public World (2nd Edition) by Charlie Kaufman, Radia Perlman, and Mike Speciner, Prentice-Hall, ISBN: 0-13-046019-2, May 2002.
A good technical book that covers hash functions, public key cryptography, digital signatures and related security technologies.

Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd Edition) by Bruce Schneier, Wiley, ISBN: 0-471-12845-7, 1996.
This book still the major reference work for core security technologies. I've heard that "RBS" still stands for "read Bruce Schneier" in some computer security circles.

DNS and BIND (5th Edition) by Cricket Liu and Paul Albitz, O'Reilly, ISBN: 978-0-596-10057-5, 2006.
By now a venerable item on the bookshelf, this book remains the best practical reference manual for BIND.

Internet Standards Documents and Drafts

RFC 3833	A Threat Analysis of the Domain Name System, D. Atkins, R. Austein, August 2004.
RFC 4033	DNS Security Introduction and Requirements, R. Arends et al, March 2005.
RFC 4034	Resource Records for the DNS Security Extensions, R. Arends et al, March 2005.
RFC 4035	Protocol Modifications for the DNS Security Extensions, R. Arends et al, March 2005.
RFC 5155	DNS Security (DNSSEC) Hashed Authenticated Denial of Existence, B. Laurie et al, March 2008.

Appendix I

Public Key Cryptography and PKIs

One of the longstanding approaches to this question of authenticity of data is that of 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 relating to the authenticity and secrecy of communications are particularly novel, and the various forms of response to these rather demanding requirements over the years form the rich and colourful history of cryptography.

A common means of assuring communications is through the use of shared 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 else intercepts the message it is intended 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.

This model of secure communication using shared keys still not very useful in the real world. It is 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 replace my keys, or you wish to replace your keys.

Some terminology that may help at this juncture: If the key is symmetric the encipher and decipher keys are 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.

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. Anyone who intercepts the message cannot make any sense of it, as they do not possess your private key. Similarly, if I encipher a message using my private key, then only my public key can decipher it. While anyone is then able to decipher the message, the value here lies in the ability to authenticate the fact that only I could've enciphered the message as I am the only person who has possession of my private key.

Now, if I use my private key to encipher a message, and then encipher the result using your public key, then I have constructed a message that only I could've generated and only you can read. And for you to read it, you need to use your private key to decipher the message, and then use my public key to decipher that message to end up with the original message. In other words such a system allows for secrecy and authenticity. Importantly, neither of us have had to use a shared secret. I only need to have knowledge of your public key, together with my secret private key, to construct the message, and you only need to have knowledge of my public key, together with your secret private key, to decipher the message. We don’t have to meet to share a secret, and I can conduct secure conversations with many different folk using my single private key and their respective public keys.

So now we know how useful such an asymmetric two way 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, $f(x)$ can be "$x + 1$" and $g(x)$ can be "$x - 1$".

Now lets add a further constraint, that knowledge of one function provides no useful clue as to the other function. Simple additive and multiplicative functions fail with this latter constraint, 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 integer 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, with a pair of functions which are the inverse of each other, but where knowledge of one function does not necessarily allow one to deduce the inverse function. In this case the function itself can be made public, as can the parameter for the initial transform, while the parameter for the inverse operation can be kept secret. Using the above function of exponentiation using modular arithmetic with a prime base, then the calculation of the inverse parameter value is similar to the calculation of prime factors. For extremely large integer values this problem will take an exceptionally long time to compute.

Back to the private / public key pairs, 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 public key, and not that of anyone else. 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?

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 feel able to 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.

This can be generalized into an approach of using chains of knowledge and role authorities to create chains of trust. If you and I both trust some third party, and this same 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, such as a "web of trust" model, or using a collective trust thresholds. The resultant structure of trust relationships and associated credential relationships, whatever its chosen form, is generally collectively referred to as a public key infrastructure, or PKI.

A PKI allows a set of parties, who adhere to a set of common operational practices and policies that collectively define the PKI, the ability to validate the association of a particular entity with a given public key value.

Appendix II

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.

The DNSKEY data (RDATA) consists of a Flags field, a Protocol value, an Algorithm Identifier and a Public Key value.

Flags:

If bit 7 is set (value is 256), then the DNSKEY record holds a DNS Zone Key, and the key can be used to verify RRSIGs that cover RRsets within the zone named as the DNSKEY RR's owner name. If bit 15 is set (value is 1 or 257), then the DNSKEY can be used as a Secure Entry Point (SEP). If the zone distinguishes between Zone Signing Keys (ZSKs) and Key Signing Keys (KSKs) as part of its key management practices, then the KSKs can be identified through the odd number value in the Flags field

Protocol:

The only defined value for this field is 3 at present. All other values invalidate the DNSKEY RR.

Algorithm:

This field identifies the algorithm associated with the Public Key value. The set of defined values is held in the IANA Registry: http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml

Public Key:

The Public Key Field holds the public key material. The format depends on the algorithm of the key being stored.

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==  )
```

In this example, 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 example, 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.

The RRSIG RR RDATA consists of a Type Covered field, an Algorithm field, a Labels field, an Original TTL field, a Signature Expiration field, a Signature Inception field, a Key Tag, the Signer's Name field, and the Signature field.

Type Covered:

The Type Covered field identifies the type of the RRset that is being signed by this RRSIG record.

Algorithm:

This field identifies the algorithm used to create the signature. The set of defined values is held in the IANA Registry: http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml

Labels:

This field specified the number of labels in the original RRSIG RR owner name. This field is used to determine if the answer has been synthesized from a wildcard label.

Original TTL:

The Original TTL field specifies the TTL of the covered RRset as it appears in the authoritative zone. The Original TTL field is necessary because a caching resolver decrements the TTL value of a cached RRset, so the TTL in a response is not necessarily the TTL that is part of the originally signed data. In order to validate a RRset signature, a validator requires the original TTL.

Signature Expiration,
Signature Inception:

These fields specify a validity period for the signature. The RRSIG record must not be used before the Signature Inception Date, nor after the Signature Expiration date.

Key Tag:

The Key Tag field in the RRSIG and DS resource record types provides a mechanism for selecting a public key, as a combination of owner name, algorithm, and key tag can efficiently identify a DNSKEY record. Both the RRSIG and DS resource records have corresponding DNSKEY records. The Key Tag field in the RRSIG and DS records can be used to help select the corresponding DNSKEY RR efficiently when more than one candidate DNSKEY RR is available.

Signer's Name:

The Signer's Name field value identifies the owner name of the DNSKEY RR that a validator should use to validate this signature. This is the name of the zone of the covered RRset.

Signature:

The Signature field contains the cryptographic signature that covers both the RRset specified by the RRSIG owner name, RRSIG class, and RRSIG Type Covered field and the the RRSIG RDATA itself.

An example RRSIG record for the zone example.com is as follows:

```
    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. Te values 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 combined RRset and the RRSIG data.

NSEC:

The DNSKEY and RRSIG records can be used to check the authenticity of a DNS response, where there is a DNS response. However an important aspect of the DNS is where the query refers to a non-existent name, or a non-existent RR. In this case there is no authoritative data to return. However, a client needs to be able to validate this "non-existent data" response, and the DNSSEC NSEC RR is intended to allow the client to confirm the authenticity of this response.

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".

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 verifiable evidence that the name, or the RR type does not exist in the authoritative zone.

For example, if the zone contained the labels "alpha" and "beta", then there would be a NSEC RR for alpha, and the RR value of this NSEC RR would be beta, indicating that there are no defined labels 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".

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 claiming that a name does not exist, whereas the NSEC record explicitly informs the client of the "gap". Note also that NSEC RRs are also signed with an associated RRSIG RR.

The NSEC RR RDATA consists of the Next Authoritative Name (in canonical order), and the set of RR types that are defined for the owner name of this RR.

An example NSEC record for the name alpha.example.com is as follows:

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

In this case the name "beta.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 defined for 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 with the NSEC3 RR.

DS:

The issue of validation of the zone public key remains unaddressed with the first three RR 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?

One approach is for every client to maintain a local set of current public keys for each signed DNS zone. This approach creates some formidable operational challenges, and is simply impractical in a scenario of comprehensive adoption of DNSSEC.

The other approach is to effectively assume that DNSSEC is universally adopted, and note that in a hierarchical delegation structure, as used by the DNS, each zone has some authoritative knowledge about the delegated child zones.

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 single unique 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 validation operation relies on the parent zone key, and the question is how can this key be validated? The same process can be applied here, using the DS RR of the parent zone. The process stops when the DNSSEC client encounters a "trusted" key. The ideal "trust key" is the DNSKEY of the root zone, but this assumes that all zones between the zone whose DNSKEY is being validated and the root zone are DNSSEC signed. In other words this interlocking of zone keys only works when DNSSEC is universally used. In the absence of such a path to the signed root zone, 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 dskey.example.com is as follows:

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

In this example the 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 of the DNSKEY owner name concatenated with the DNSKEY RDATA, which is, in turn, the flags, protocol, algorithm and public key public key of the DNSKEY.

NSEC3:

The zone enumeration issue is a significant problem for many DNS zones, and the zone administrators found the zone enumeration capability of NSEC RRs to be unacceptable. By carefully constructing a sequence of requests for non-existent names, it is possible to construct a complete set of defined zone names and the associated set of defined RRsets for each name, all from the DNSSEC NSEC responses.

NSEC3 is an alternative to the NSEC RR. NSEC3 defines an ordering of the names in the zone that uses a hashed space to report intervals of non-existent names. This makes it extremely difficult to enumerate all the defined names in the space. However, a client can still walk the zone, though the "walk" is in effect a random search, as each probe has to be at some new randomly selected place in the zone, and not based on the previous NSEC3 response. For most purposes, the challenge posed by the effort to perform a complete enumeration of the zone using this random query pattern for NSEC3 records is considered to offer sufficient protection of the information in the zone.

Appendix III

A DNSSEC Worked Example

Signing a Zone

The following is an example walk-through of the creation of a DNSSEC-signed zone using the suite of tools provided wit release 9.7.1 of the DNS BIND software from ISC (http://www.isc.org). I'll use as a guide for this worked example the "DNSSEC HOWTO" tutorial by Olaf Kolkman (http://www.nlnetlabs.nl/dnssec_howto/dnssec_howto.pdf)

The zone we'll sign here is `dnssec.potaroo.net`.

The first step is to ensure that the zone's servers are capable of supporting DNSSEC. The way to achieve this will vary according to the DNS Server software being used.

In the case of the BIND server, the essential first step is to compile the software with *openssl*, and enabling the use of DNSSEC with the "`dnssec-enable yes;`" directive as part of the `options` specified in `named.conf`.

```
$ cat /etc/namedb/named.conf
…
options {
      …
      dnssec-enable yes;
      …
      };
```

The next step is key creation. Common operational practice today is to use two distinct key functions: the *key-signing keys* (KSKs) as the anchor point of the chain of authority to the data that needs to be validated, and *zone-signing keys* (ZSKs) that are used to sign the zone data.

In the case of BIND, this can be achieved using the `dnssec-keygen` command, creating a zone signing key, then a key signing key.

```
$ dnssec-keygen -r /dev/random -a RSASHA1 -b 1024 -n ZONE dnssec.potaroo.net
Kdnssec.potaroo.net.+005+16482

$ dnssec-keygen -r /dev/random -f KSK -a RSASHA1 -b 1280 -n ZONE dnssec.potaroo.net
Kdnssec.potaroo.net.+005+64273
```

The public key parts of these key pairs are added to the zone to be signed.

```
$ cat dnssec.potaroo.net
;
; example dnssec-signed zone
;
$TTL        1d
$ORIGIN     dnssec.potaroo.net.
@           IN SOA    dns0.potaroo.netnet. gih.apnic.net. (
            2010062101      ; Serial
            3h        ; Refresh
            1h        ; Retry
            1w        ; Expire
            3h )            ; Neg. cache TTL

;
;           name servers
;
```

```
           NS dns0.potaroo.net.
;
;          zone contents
;
bogong  A       203.133.248.6
bogong  AAAA    2401:2000:6660::6
wally   A       202.158.221.222
wally   AAAA    2001:388:1:4007:20e:cff:fe4b:f987
;
;          zone keys
;
$include  Kdnssec.potaroo.net.+005+64273.key     ; KSK
$include  Kdnssec.potaroo.net.+005+16482.key     ; ZSK
```

The zone is then signed, using the `dnssec-signzone` command.

```
$ dnssec-signzone -r /dev/random -o dnssec.potaroo.net \
                  -k Kdnssec.potaroo.net.+005+64273 \
                  dnssec.potaroo.net  Kdnssec.potaroo.net.+005+16482.key

Verifying the zone using the following algorithms: RSASHA1.
Zone signing complete:
Algorithm: RSASHA1: KSKs: 1 active, 0 stand-by, 0 revoked
                    ZSKs: 1 active, 0 stand-by, 0 revoked
dnssec.potaroo.net.signed
```

The signed zone is as follows:

```
$ cat dnssec.potaroo.net.signed
; File written on Mon Jun 21 02:49:13 2010
; dnssec_signzone version 9.7.1
dnssec.potaroo.net.  86400  IN SOA dns0.potaroo.netnet. gih.apnic.net. (
                                   2010062101 ; serial
                                   10800      ; refresh (3 hours)
                                   3600       ; retry (1 hour)
                                   604800     ; expire (1 week)
                                   10800      ; minimum (3 hours)
                                   )
                     86400  RRSIG  SOA 5 3 86400 20100721014913 (
                                   20100621014913 16482 dnssec.potaroo.net.
                                   r/xnQf+ZwDEyMVCIfy1yCFNTl+OF5qWjchYm
                                   dr//w9wOn9H/zrSjDNs8Bqygi2Hk3XXy+AU5
                                   zWStag4ujMdQJD9qEsYSbXafVlI+K+s/Aypd
                                   xKUDVh1LJdQSGIURx7smmg1+s+rv4jNLX3ym
                                   JSOyeDkb3BMOfikwbevTUjrqOB8= )
                     86400  NS     dns0.potaroo.net.
                     86400  RRSIG  NS 5 3 86400 20100721014913 (
                                   20100621014913 16482 dnssec.potaroo.net.
                                   IvLpRI8ck480vD71w0865NL/xB8zuRPdugHD
                                   MEV8IpmAN9HHQyjTIt/VPddVeFhYYGCJr9a+
                                   aRFdw7ySKJnJX96AOgSLRLI4yUmeA1ZeY9Rk
                                   u47PmqLZSYiz88KFu7tE8f8PM9cdf8PJA7+N
                                   bDzLjMFXFTVtwn9Vk9rbZmLsEyM= )
                     10800  NSEC   bogong.dnssec.potaroo.net. NS SOA RRSIG NSEC DNSKEY
                     10800  RRSIG  NSEC 5 3 10800 20100721014913 (
                                   20100621014913 16482 dnssec.potaroo.net.
                                   eNKlEj4mNN0jMQ4vVhPgQFCLXtkQ1jsjiqF6
                                   DYQkIlA7xCbo0dGKkdz8aHPg2aAZy19UpLpz
                                   hTJSart+0RMX7nVEBCgBaM0zGYdHi4ZPeKxk
                                   R192VneP12d04xVKA8kKmZk+yBw8AcfWMneC
                                   hStzqMOLlc4Rcu30LQO1zGqtHc8= )
                     86400  DNSKEY 256 3 5 (
                                   AwEAAbGuQRpZhiQd61wcv+MmTz4AuBrPWs7f
                                   o7q2HiW8rwHnBOknPnI1vGkCjLDJoW8zN8jh
                                   /wiIpuvFtjI2/4Sz2LMPWoQUewojlzaTkYje
                                   I6GkeMGpvS5Fnjk6YRxfzq8+COFMWXElQLwS
                                   mwUZYmsoQoOafkUKLRPtbTScbdTobGfX
                                   ) ; key id = 16482
                     86400  DNSKEY 257 3 5 (
                                   AwEAAdmfkEMaaUrwh3+vxkTlvcn+2KJLC/A0
                                   jflAonVEc06PohnYBA8pA+v500EJrMhATLwa
                                   LZF86qklr+8d6SgG5f0XU8IPugM3GTCuLkkK
                                   mYjzw5uJANAekUMlFNfezTzKNF6OMQxG11++
                                   T7rbMEGmn/L3jGmVD9Ro5qpqpPUm8EBwLXS
```

```
                                  2ALIA3MVSRUS5Hbb12x/t19DQlAQLVZC07/q
                                  y3E=
                                  ) ; key id = 64273
                    86400  RRSIG  DNSKEY 5 3 86400 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  qaQitL19IPGaEOMeDHqRy1Tz3MrLHWzc5oYz
                                  298AUOOKNfpLU8aemck4pJCPoj3h2TiRVmJg
                                  CgJFV/sxaFALsi35sg18agTRGfDbskHGXchz
                                  eSJWBrepxnu3GUkKhji/lUB1Vy1ns4fXkQpf
                                  HPMja4kFAnhd/lRjR85tcKOlk7I= )
                    86400  RRSIG  DNSKEY 5 3 86400 20100721014913 (
                                  20100621014913 64273 dnssec.potaroo.net.
                                  sUynXvlfGfSNcLHiv8e8wvNHbpurhsnd+FcW
                                  wOwcW0tFPL5RtxcS/v5WEt21Ny1vkaw8JtSY
                                  26dJ72pvpav6EZJGczYzwWzFu8rLDWnB1gI8
                                  pzrAPri1gTNbrWJoRl61IaPq/5ip3bGVDUFM
                                  CSFTgIU6O2rTuD9CQn5Ox1xffxvB3ATsJHyg
                                  +b5tcHdYIGT7My4ELbdc0nn2BYuQAGjx0g== )
wally.dnssec.potaroo.net. 86400   IN A   202.158.221.222
                    86400  RRSIG  A 5 4 86400 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  FSudndyQLU+Zr54GT6yM2dp1h/dM9OL1PmxC
                                  fGyY6F6n7YtUpHR/xItKGkeqy0NJ/8SfgRX0
                                  avGgug64vheyS1dwQdiiqlyjue6C63by8Owd
                                  xeSirKmYbt5/FftrzXqqBeQdE6lArNheaN4d
                                  l0lRT1lEC/SUBjb2w45aYzIo/cA= )
                    86400  AAAA   2001:388:1:4007:20e:cff:fe4b:f987
                    86400  RRSIG  AAAA 5 4 86400 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  KJartXEl6FgeIzeQ6SXeGhibmaeSygMjMC7N
                                  HbMEc+MHp8nMAeRvKk4J59xwzSduLHUbUTBV
                                  MsoviBAk7p2FoZ/JNlY5DIFFNPxoHJa6X+st
                                  Kprhu1SgOnYQxP0CyGiMZpcEC7RWK/MVUnQ0
                                  UsbJ0vHTEjGbpHP2gR/M9JLUfwc= )
                    10800  NSEC   dnssec.potaroo.net. A AAAA RRSIG NSEC
                    10800  RRSIG  NSEC 5 4 10800 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  QgAF2Bnlvf1g+rhvUi1RJ36lnT/AjLO2ikBt
                                  Pn8DU2XvJHXjoP2mYdXwN9yM7DegteXe1YXe
                                  sTEFcCddSWnbX83ccanmTtr12RRac72GTSPf
                                  ataM3dnpw7JHrT6JbvUkXtza82lmMp0lYLWs
                                  ZL0X92Rxo6cG+BLrEoJrx8DG4qg= )
bogong.dnssec.potaroo.net. 86400 IN A   203.133.248.6
                    86400  RRSIG  A 5 4 86400 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  HPExYnOugiydDe3VqlECPilDt9zzQe1/sbh7
                                  jfrVVih7FyOvMl63Nb8XL7clrXUy+m2L0XUV
                                  2w055Wbb5YhVRXHgKprokQsgo18BnhDzus+Q
                                  9zef4tGv4IT41OTLUbwrsV5ciJ4THhSBjpkk
                                  X3Em9kX7lYK4xUh3bdnb0O3VosE= )
                    86400  AAAA   2401:2000:6660::6
                    86400  RRSIG  AAAA 5 4 86400 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  C/u5R83btmVi3Rm9PENDyUDWWYn99mFvBcMR
                                  DenkLDBv/+JJMBsyw+zj7Ift/GGM8KTrTBjS
                                  9DCeA0XVMvZMoYhzM9cAX3eFG01qxDUEi0tK
                                  OllPdfxCXs7JvigvZbtLTgNaejHWXvdwDAwO
                                  6lvMAhswVnJ+uLloiSW1uH87YI8= )
                    10800  NSEC   wally.dnssec.potaroo.net. A AAAA RRSIG NSEC
                    10800  RRSIG  NSEC 5 4 10800 20100721014913 (
                                  20100621014913 16482 dnssec.potaroo.net.
                                  KUDpCQ7a27Zuu8LWvK+1rMbZKdzx3AILaEWk
                                  ceBFI/iqHZm7BNO1tdIi25uhPzp9kla9wUf1
                                  Y7qVQvIJ2MBy5ZJMwgxLXvcWs+emq9kzBVBe
                                  5Gh52W34LCo+B2rJ+xSB6qLEVjbNnr4tancb
                                  t4CsezFvupSVU6mBQC0NsWiTeIM= )
```

The signed zone is added to the set of served zones, and the BIND server configuration is reloaded.

```
$ tail -5 named.conf

zone "dnssec.potaroo.net" {
        type master;
        file "master/dnssec.potaroo.net.db" ;
        };

$ kill -HUP `cat /var/run/named/pid`
```

The KSK is added to the set of local trust keys for a DNSSEC-aware resolver:

```
$ cat named.conf
…
trusted-keys {
…
"dnssec.potaroo.net."        257 3 5
"AwEAAdmfkEMaaUrwh3+vxkTlvcn+2KJLC/A0jflAonVEc06PohnYBA8pA+v500EJrMhATLwaLZF86qklr+8d
6SgG5f0XU8IPugM3GTCuLkkKmYjzw5uJANAekUMlFNfezTzKNF6OMQxG11++T7rbMEGmn/L3jGmVD9Ro5qpqp
pPUm8EBwLXS2ALIA3MVSRUS5Hbb12x/t19DQlAQLVZC07/qy3E=";
…
};
```

This resolver can be queried:

```
$ dig +dnssec +multiline DNSKEY dnssec.potaroo.net

; <<>> DiG 9.7.1 <<>> +dnssec +multiline DNSKEY dnssec.potaroo.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42734
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;dnssec.potaroo.net.       IN DNSKEY

;; ANSWER SECTION:
dnssec.potaroo.net. 86400 IN DNSKEY     256 3 5 (
                          AwEAAbGuQRpZhiQd61wcv+MmTz4AuBrPWs7fo7q2HiW8
                          rwHnBOknPnI1vGkCjLDJoW8zN8jh/wiIpuvFtjI2/4Sz
                          2LMPWoQUewojlzaTkYjeI6GkeMGpvS5Fnjk6YRxfzq8+
                          COFMWXElQLwSmwUZYmsoQoOafkUKLRPtbTScbdTobGfX
                          ) ; key id = 16482
dnssec.potaroo.net. 86400 IN DNSKEY     257 3 5 (
                          AwEAAdmfkEMaaUrwh3+vxkTlvcn+2KJLC/A0jflAonVE
                          c06PohnYBA8pA+v500EJrMhATLwaLZF86qklr+8d6SgG
                          5f0XU8IPugM3GTCuLkkKmYjzw5uJANAekUMlFNfezTzK
                          NF6OMQxG11++T7rbMEGmn/L3jGmVD9Ro5qpqppPUm8EB
                          wLXS2ALIA3MVSRUS5Hbb12x/t19DQlAQLVZC07/qy3E=
                          ) ; key id = 64273
dnssec.potaroo.net. 86400 IN RRSIG DNSKEY 5 3 86400 20100721014913 (
                          20100621014913 16482 dnssec.potaroo.net.
                          qaQitL19IPGaEOMeDHqRy1Tz3MrLHWzc5oYz298AUOOK
                          NfpLU8aemck4pJCPoj3h2TiRVmJgCgJFV/sxaFALsi35
                          sg18agTRGfDbskHGXchzeSJWBrepxnu3GUkKhji/lUB1
                          Vy1ns4fXkQpfHPMja4kFAnhd/lRjR85tcKOlk7I= )
dnssec.potaroo.net. 86400 IN RRSIG DNSKEY 5 3 86400 20100721014913 (
                          20100621014913 64273 dnssec.potaroo.net.
                          sUynXvlfGfSNcLHiv8e8wvNHbpurhsnd+FcWwOwcW0tF
                          PL5RtxcS/v5WEt21Ny1vkaw8JtSY26dJ72pvpav6EZJG
                          czYzwWzFu8rLDWnB1gI8pzrAPri1gTNbrWJoRl61IaPq
                          /5ip3bGVDUFMCSFTgIU6O2rTuD9CQn5Ox1xffxvB3ATs
                          JHyg+b5tcHdYIGT7My4ELbdc0nn2BYuQAGjx0g== )

;; Query time: 205 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Mon Jun 21 07:30:11 2010
;; MSG SIZE  rcvd: 763
```

The "ad" flag in the response indicates that the response was authenticated using DNSSEC. The local resolver's log also shows this local validation result:

```
$ more named-dnssec.log

21-Jun-2010 07:30:11.207 dnssec: debug 3: validating @0x8028f6b00: dnssec.potaroo.net
DNSKEY: starting
21-Jun-2010 07:30:11.207 dnssec: debug 3: validating @0x8028f6b00: dnssec.potaroo.net
DNSKEY: attempting positive response validation
21-Jun-2010 07:30:11.207 dnssec: debug 3: validating @0x8028f6b00: dnssec.potaroo.net
DNSKEY: verify rdataset (keyid=64273): success
21-Jun-2010 07:30:11.207 dnssec: debug 3: validating @0x8028f6b00: dnssec.potaroo.net
DNSKEY: signed by trusted key; marking as secure
```

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He 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 was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of Trustees of the Internet Society from 1992 until 2001.

www.potaroo.net