The ISP Column
A monthly column on things Internet

October 2016

Geoff Huston

ECDSA and DNSSEC

Two years ago I reported on the use of the elliptical curve cryptographic algorithm in generating digital signatures for securing the DNS (DNSSEC) (http://www.potaroo.net/ispcol/2014-10/ecdsa.html). The conclusion at the time was hardly encouraging:

"Will ECDSA ever be useful tool for DNS and DNSSEC? As good as ECDSA is in presenting strong crypto in a smaller number of bits, it is still an alien algorithm for much of today's Internet. So, sadly, I have absolutely no idea how to answer that question as to when it may become genuinely useful for use in DNSSEC."

Let's see what's changed over the past 24 months.

Erasthosthenes of Cyrene was an ancient Greek of some astounding learning, becoming the Chief Librarian at that fabled wonder of the ancient world, the library of Alexandria, in the third century BCE. Not only has he been credited as the first person to calculate the circumference of the earth, but for the purposes of this article his claim for posterity is based on his work in number theory, and in particular his invention of the Sieve of Erastosthenes as a means of identifying prime numbers.

His method is complete and accurate, but for very large numbers the process is inordinately slow. And we haven't really made it much faster in the intervening 2,200 years. The related problem, that of computing the prime number factors of a very large integer also takes a long time as at its heart is a basic enumeration problem that grows as the size of the number grows. It's not an impossible problem, but even with the most powerful computers available to us today, it may take months, or years, to perform this operation depending on the size of the number. It is on this foundation that we have constructed much of the security infrastructure of today's computers and the Internet itself. The widely used RSA algorithm uses this principle of the difficulty of prime number factorization as its cornerstone.

The basic principle behind RSA is the observation that it is relatively fast to find three very large positive integers e, d and n such that with modular exponentiation for all m:

$$(m^e)^d \equiv m \pmod{n}$$

but reverse engineering this relationship, and in particular, even when the values of e, n (and even m) are known, finding the value of d can be computationally far more expensive than generating the three numbers in the first place. In terms of order of magnitude, its computational expense is not dissimilar to the problem of of prime number factorization. This leads to the ability to generate a code that is relatively fast to generate and extremely difficult to break.

This relationship leads to an asymmetric key relationship that can be used to construct a complementary key pair where a code generated by one key can only be decoded by its complementary key.

Given a public key of the form (n, e), and a message m, the message can be encrypted to a cyphertext c using:

$$c \equiv m^{\mathbf{e}} \pmod{\mathbf{n}}$$

The complementary private key is (n, d) and it can be used to decode the message through computing

$$c^{\mathbf{d}} \equiv (m^{e})^{\mathbf{d}} \equiv m \pmod{\mathbf{n}}$$

At the heart of this relationship is the value of n, which is the product of two (large) prime numbers. e is often set to the value $2^{16}+1$ (65537).

Of course what is "computationally expensive" is a relative term depending on when you make the observation. Tasks that were computationally infeasible a couple of decades ago may well be commonplace tasks using current computing capabilities. This means that as computing capabilities improve over time, the size of the numbers used in RSA encryption need to increasing in size.

"As of 2010, the largest factored RSA number was 768 bits long (232 decimal digits). Its factorization, by a state-of-the-art distributed implementation, took around fifteen hundred CPU years (two years of real time, on many hundreds of computers). No larger RSA key is publicly known to have been factored. In practice, RSA keys are typically 1024 to 4096 bits long. Some experts believe that 1024-bit keys may become breakable in the near future or may already be breakable by a sufficiently well-funded attacker (though this is disputed); few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if *n* is sufficiently large. If *n* is 300 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. Keys of 512 bits have been shown to be practically breakable in 1999 when RSA-155 was factored by using several hundred computers and are now factored in a few weeks using common hardware. Exploits using 512-bit code-signing certificates that may have been factored were reported in 2011. A theoretical hardware device named TWIRL and described by Shamir and Tromer in 2003 called into question the security of 1024 bit keys. It is currently recommended that *n* be at least 2048 bits long."

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

So for secure systems that use RSA as their cryptographic algorithm, we've seen pressure to continuously increase the size of the keys. This is true across the entirety of application of RSA-based cryptography, including the area of security in the DNS.

Keys Lengths in the Root Zone of the DNS

When the root zone of the DNS was first sized in 2010, the system used a conventional split across two key pairs: a stable "Key-Signing Key" (KSK) that is used to sign just the DNSKEY record of the root zone, and a "Zone-Signing Key" (ZSK) that is used to generate all the other signatures in the signed root zone. The KSK uses a 2048 bit key, and until now (October 2016) the ZSK was set to 1024 bits in size.

Across the coming months the ZSK will be changed in size to 2048 bits in length, largely in response to the general view that best practice for RSA keys is to be at least 2048 bits in length. (the detailed steps involved in this change in the size of the Root Zone ZSK is described in a June 2016 presentation at NANOG by Duane Wessels:
https://www.nanog.org/sites/default/files/Wessels02.pdf)

But if 2048 bits is good, isn't 3072 bits better? Why don't we all just use massive keys in RSA all of the time? One constraint is the cost to generate the keys. The larger the key size the higher the computational load to generate the keys and the encrypted payload. At some point increasing the key size increases the load on the encryptor without substantially altering the security properties of the result. But in some areas of application there are very real size constraints and supporting arbitrarily long RSA keys is considered infeasible irrespective of the cost of generating the keys in the first place.

The DNS is one of those bounded areas.

The original DNS specification used a maximum payload of 512 octets for UDP messages (RFC 1035). DNS servers were required to truncate longer responses, and a DNS client receiving a truncated response was expected to requery using TCP.

A subsequent extension to the DNS protocol, EDNS(0) (RFC6891), allowed a client to specify that it could handle a larger UDP response size than 512 octets, and in theory at any rate it might be possible for a client to assert that it could handle up to 65,535 octets in size, and if UDP and TCP are able to interpret size fields as specified in RFC2675, then in IPv6 a jumbogram could be up to 4 billion octets in size. But of course theory and practice can diverge markedly, and while the IP and transport level protocols can be coerced into supporting arbitrarily large datagrams, the practice is far more mundane. IP fragments work in a haphazard manner in IPv4 and the story in IPv6 is far more dismal than that. If you want to reliably receive large blocks of information across any network path, where "large" is any number greater than 1,240 octets, over either IPv4 of IPv6, then datagrams are not the answer! If you are tolerant of some loss then 1,500 octets is the next boundary point, as larger packets typically require IP level fragmentation, and at that point the packet loss rate starts to rise sharply.

What can we do for the digital signatures used in DNSSEC? If we want to continue to use RSA as the cryptographic algorithm, then we need to look at deploying ever larger key sizes, and performing key rollovers with ever larger DNS response packets. If IP packet fragmentation presents unacceptable loss rates, then this implies that we need to look at the evolution of DNS from datagram transactions into one that uses a streamed transport protocol, such as TCP. This is a statement not without its implications. We have grown used to the DNS operating in a lightweight datagram transaction model. There are some concerns that we might not be able to support the same performance profile for the DNS, and operate within the same cost and service parameters if we have to use TCP for all DNS transactions.

But that's not the only option we have. Another response is to try and reduce the size of the DNS response by changing the crypto algorithm, and it's here that ECDSA has something to offer.

What is ECDSA?

Briefly, ECDSA is a digital signing algorithm that is based on a form of cryptography termed "Elliptical Curve Cryptography". This form of cryptography is based on the algebraic structure of elliptic curves over finite fields.

The security of ECC depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points. This is

> phrased as a discrete logarithm problem, solving the equation $b^k = g$ for an integer k when b and g are members of a finite group. Computing a solution for certain discrete logarithm problems is believed to be difficult, to the extent that no efficient general method for computing discrete logarithms on conventional computers is known. The size of the elliptic curve determines the difficulty of the problem.
>
> This approach is in contrast to the widely used RSA algorithm, that uses exponentiation modulo a product of two very large primes, to encrypt and decrypt. The security of RSA is based on a somewhat different assumption of difficulty, namely the assumption of the difficulty of prime factoring large integers, a problem for which there is no known efficient general technique.

The major attraction of ECDSA is not necessarily in terms of any claims of superior robustness of the algorithm as compared to RSA approaches, but in the observation that Elliptic Curve Cryptography allows for comparably difficult problems to be represented by considerably shorter key lengths. If the length of the keys being used is a problem, then maybe ECC is a possible solution.

> As pointed out in RFC6605:
>
> "Current estimates are that ECDSA with curve P-256 has an approximate equivalent strength to RSA with 3072-bit keys. Using ECDSA with curve P-256 in DNSSEC has some advantages and disadvantages relative to using RSA with SHA-256 and with 3072-bit keys. ECDSA keys are much shorter than RSA keys; at this size, the difference is 256 versus 3072 bits. Similarly, ECDSA signatures are much shorter than RSA signatures. This is relevant because DNSSEC stores and transmits both keys and signatures."
>
> RFC6605, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", P. Hoffman, W.C.A. Wijngaards, April 2012

We are probably right to be concerned over ever-expanding key sizes in RSA, and the associated implications of the consequent forced use of UDP fragments for the DNS when packing those longer key values into DNSSEC-signed responses. If UDP fragmentation in the DNS is unpalatable, then TCP for the DNS probably not much better, given that we have no clear idea of the scalability issues in replacing the stateless datagram transaction model of the DNS with that of a session state associated with each and every DNS query. The combination of these factors make the shorter key values in ECDSA a pretty attractive algorithm for use in DNSSEC.

If it's so attractive, then why aren't we all using ECDSA already?

Well, there's one very relevant question that should be answered before you all head off to use ECDSA to sign your DNS zones. Is the ECDSA algorithm as widely supported by DNSSEC-Validating resolvers as the RSA algorithms?

> There are reasons why we should ask this question. Elliptical Curve Cryptography is not without its elements of controversy. As Wikipedia explains:
>
> "In 2013, the New York Times stated that Dual Elliptic Curve Deterministic Random Bit Generation (or Dual_EC_DRBG) had been included as a NIST national standard due to the influence of NSA, which had included a deliberate weakness in the algorithm and the recommended elliptic curve. RSA Security in September 2013 issued an advisory recommending that its customers discontinue using any software based on Dual_EC_DRBG. In the wake of the exposure of

> Dual_EC_DRBG as "an NSA undercover operation", cryptography experts have also expressed concern over the security of the NIST recommended elliptic curves, suggesting a return to encryption based on non-elliptic-curve groups."
> http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

A similar perspective on Dual_EC_DRBG was the topic of an earlier 2007 essay by Bruce Schneier:

> "If this story leaves you confused, join the club. I don't understand why the NSA was so insistent about including Dual_EC_DRBG in the standard. It makes no sense as a trap door: It's public, and rather obvious. It makes no sense from an engineering perspective: It's too slow for anyone to willingly use it. And it makes no sense from a backwards-compatibility perspective: Swapping one random-number generator for another is easy. My recommendation, if you're in need of a random-number generator, is not to use Dual_EC_DRBG under any circumstances. If you have to use something in SP 800-90, use CTR_DRBG or Hash_DRBG. In the meantime, both NIST and the NSA have some explaining to do."
> https://www.schneier.com/essay-198.html

But let me hasten to note that Dual_EC-DRBG is not ECDSA P-256, and no such suspicions of exposure have been voiced for ECDSA P-256 or its related cousin ECDSA P-384. However, there is still a lingering perception issue with certain variants of ECC random bit generation, even though does not mean that it is justified to believe that all Elliptical Curve Cryptography is similarly tainted with suspicion.

If we are able to discount such suspicions, and assert that ECDSA P-256 and ECDSA P-384 are robust in terms of cryptographic integrity, are there any other problems with the use of ECDSA?

ECDSA has a background of patents and IPR claims, particularly, but not entirely, associated with the entity Certicom, and for some time this IPR confusion was considered sufficient reason for many distributions of crypto libraries not to include ECDSA support (http://en.wikipedia.org/wiki/ECC_patents). OpenSSL, the most widely adopted open crypto library, added ECDSA from version 0.9.8 in 2005, but a number of software distributions took some further time to make the decision that it was appropriate to include ECDSA support (such as Red Hat Fedora, where the distribution's inclusion of ECDSA support was apparently delayed until late 2013, probably due to these IPR concerns: https://bugzilla.redhat.com/show_bug.cgi?id=319901)

Taking all this into account, it's not surprising that folk have been cautious with their approach to ECDSA, both to use it as a signing algorithm and to support its use in various crypto validation libraries.

Is that caution still lingering, or are we now confident in using ECDSA on the Internet?

The ECDSA Question

The document describing the use of ECDSA as a DNSSEC signing algorithm is RFC6605, published in 2012. This document standardised a profile for use of ECDSA P-256 and ECDSA P-384. The question posed here is: what proportion of the Internet's end users use DNS resolvers that are capable of handling objects signed using the ECDSA protocol, as compared to the level of support for the RSA protocol?

At APNIC Labs, we've been continuously measuring the extent of deployment of DNSSEC for a couple of years now. The measurement is undertaken using an online advertising network to pass the user's browser a very small set of tasks to perform in the background that are phrased as the retrieval of simple URLs of invisible web "blots". The DNS names loaded up in each ad impression are unique, so

that DNS caches do not mask out client DNS requests from the authoritative name servers, and the subsequent URL fetch (assuming that the DNS name resolution was successful) is also a uniquely named URL so it will be served from the associated named web server and not from some intermediate web proxy.

The DNSSEC test uses three URLs:

- a *control* URL using an unsigned DNS name,
- a *positive* URL, which uses a DNSSEC-signed DNS name, and
- a *negative* URL that uses a deliberately invalidly-signed DNS name.

A user who exclusively uses DNSSEC validating resolvers will fetch the first two URLs but not the third (as the DNS name for the third cannot be successfully resolved by DNSSEC-validating resolvers, due to its corrupted digital signature). The authoritative name servers for these DNS names will see queries for the DNSSEC RRs (in particular, DNSKEY and DS) for the latter two URLs, assuming of course that the DNS name is unique and therefore is not held in any DNS resolver cache). This test uses the RSA algorithm.

To test the extent to which ECDSA P-256 is supported we added two further tests to this set, namely:

- a validly signed URL where the terminal zone is signed using ECDSA P-256 (protocol number 13 in the DNSSEC algorithm protocol registry), and
- a second URL where the ECDSA P-256 signature is deliberately corrupted.

What do these validating resolvers do when they are confronted with a DNSSEC-signed zone whose signing algorithm is one they don't recognize? RFC4035 provides an answer this this question:

> If the resolver does not support any of the algorithms listed in an authenticated DS RRset, then the resolver will not be able to verify the authentication path to the child zone. In this case, the resolver SHOULD treat the child zone as if it were unsigned.
>
> RFC4035, "Protocol Modifications for the DNS Security Extensions", R. Arends, et al, March 2005.

A DNSSEC-validating DNS resolver that does not recognize the ECDSA algorithm should still fetch the DS record of the parent zone, but will then complete the resolution function as if the name was unsigned, and return the resolution response. We should expect to see a user who uses such DNS resolvers to fetch the DS records for both names, but then fetch the web objects for both of these URLs as well.

A ECDSA-aware DNSSEC-validating resolver should fetch both the DS records of the parent zone and the DNSKEY record of the zone, and as the resolver will return SERVFAIL for the invalidly signed name, only the validly signed name will result in a web fetch of the corresponding URL.

ECDSA Validation Results

What we get back from the ad placement system in this experiment is a set of queries logged at the authoritative name server for the DNS names used in the experiment, and the set of fetches of the named object from the experiment's web servers.

The DNS is one of the best examples I can think of as a meta-stable, non-deterministic, chaotic system that still, surprisingly, manages to operate in a manner that appears to be largely stable and in a highly efficient manner. In the context of analysing the data generated by the experiment's DNS servers, the

linkage between the user's DNS resolution subsystem being presented with a DNS name to resolve and the queries that are presented at the experiment's authoritative name servers are often not entirely obvious. The user's local configuration may have multiple resolvers, and when the initial query does not elicit a response within some local timeout, the query will be repeated. Additional resolvers will be enlisted to assist in the resolution of the name. When a recursive resolver receives such a query it too will have its own repeat timers. And of course there are server farms and other forms of query fanout and both amplify a query and reshuffle a sequence of related queries. The result is that a single user resolution task on an uncached name may result in a large set of queries at the authoritative name server, and the order in which the queries arrive at the authoritative server may not necessarily correspond to any particular ordering of query tasks on the user's original resolver.

However, within all this induced noise there are a number of strong components of signal that points to the actions of a DNSSEC-validating resolver. A distinguishing aspect of such a resolver is that it will query for the zone's keys during the validation phase. We can use the profile of DNSKEY and DS queries as one indicator of the extent to which DNS resolvers are performing DNSSEC validation, and whether or not the resolvers recognize the ECDSA algorithm.

Over 5 days at the start of October 2016 the ad network successfully completed 25,852,032 experiments that performed the embedded set of five tests, spread across the complete span of the Internet. Of these experiments, 3,871,834 instances queried for the DNSKEY and DS records of the RSA-signed domain names and fetched the validly signed web object and did not fetch the invalidly signed web object, which is consistent with the exclusive use of DNSSEC-validating resolvers that recognize the RSA crypto algorithm. This corresponds to 14.9% of all presented experiments.

For ECDSA support, the numbers are smaller. Some 3,036,234 instances performed the same set of actions that are consistent with exclusive use of DNSSEC-validating resolvers that recognize the ECDSA P-256 crypto algorithm, or 11.74% of all presented experiments.

DNSSEC Query Profile	
Experiments	25,852,032
RSA DNSKEY	3,871,834
ECDSA DNSKEY	3,036,234

Table 1: DNS Query Profile – October 2016

These figures suggest that of the subset of users who exclusively use RSA-aware DNSSEC-Validating resolvers, one fifth or them, or 22% of these users, are using DNS resolvers that do not appear to recognize the ECDSA P-256 crypto algorithm, while the remainder are capable of handling ECDSA.

Is ECDSA Viable for DNSSEC Today?

Returning to the original question that motivated this study, is ECDSA a viable crypto algorithm for use in DNSSEC today?

The picture has changed since we originally looked at the uptake of ECDSA support in DNSSEC-validating resolvers in 2014. While it looked then that there was just insufficient support of ECDSA in validating resolvers to make the use of ECDSA viable, the picture appears to change changed considerably, and we now appear to be in a position to use ECDSA without the expectation of any significant impairment of the security properties of DNSSEC for end users.

This is an important topic for DNSSEC, as it does appear that the smaller size of ECDSA cryptographic material when compared to equivalent RSA key sizes offers us the potential to continue to use a datagram model of DNSSEC queries and responses for some time. To allow us to track this is

some detail we intend to run these measurements for an extended period and undertake continual reporting.

A 'heat map" of the state of ECDSA report is shown below:

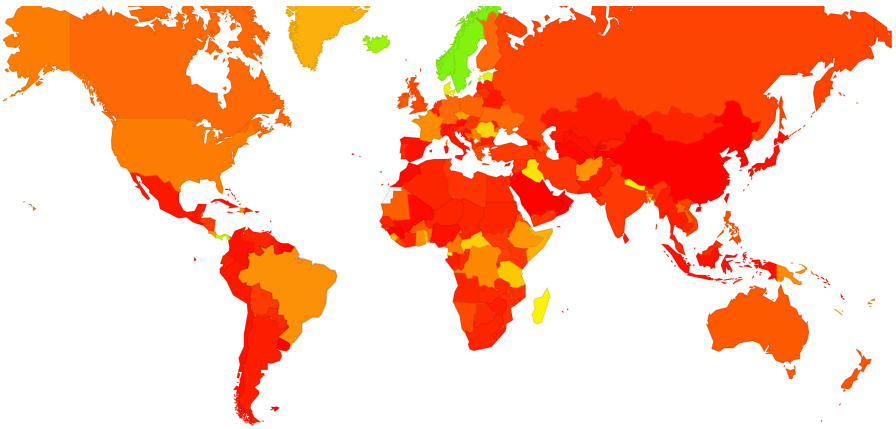

Figure 1 – Heat Map of DNSSEC Validation using ECDSA

It is also possible to identify those areas where there is a high incidence of DNS resolvers that do not recognise the ECDSA algorithm:

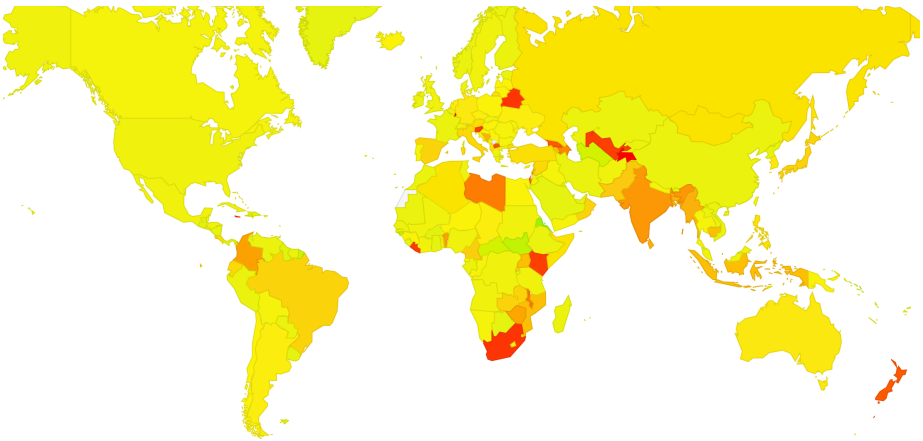

Figure 2– Heat Map of DNSSEC Validation using ECDSA : RSA ratio

These ECDSA deployment reports (and much more) can be found at the URL http://stats.labs.apnic.net/ECDSA

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 building the Internet within the Australian academic and research sector in the early 1990's. 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 and chaired a number of IETF Working Groups. He has worked as an Internet researcher, as an ISP systems architect and a network operator at various times.

www.potaroo.net

Disclaimer