The ISP Column
A column on various things Internet

August 2018
Geoff Huston

Measuring ECDSA in DNSSEC – A Final Report

Back in 2014 I wrote 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 a 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."

Four years later, let's see if we can provide an updated answer this question and hopefully put the matter to rest.

Public key cryptography relies on the construction of a particular class of whole number problems where the problem is relatively simple to construct, but incredibly challenging to solve quickly. One of the better-known examples of this class of problems is that of discovering the prime number factors of certain very large composite numbers, and this particular problem is the foundation of the most common cryptographic algorithm in use in the Internet today.

RSA Cryptography

Erastosthenes of Cyrene was a 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 enumeration 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 formed by multiplying two prime numbers, can also take a long time because at its heart lurks the same basic enumeration algorithm 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 many years to perform this operation depending on the size of the number.

For practical purposes we'll ignore the disruptive quantum computing storm clouds on the horizon that lurk in our future! That's another topic for another day!

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 (Rivest–Shamir–Adleman) algorithm uses this principle of the difficulty of prime number factorization as its corner stone. The basic principle behind RSA is the observation that it is relatively efficient for a computer to find three very large positive integers e, d and n such that with modular exponentiation for all m:

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

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 these three numbers in the first place. In terms of order of magnitude, its computational expense is not dissimilar to the problem of prime number factorization. This leads to the ability for computer algorithms to generate a public key cipher 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:

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

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

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

If the original message **M** has been *padded* to produce the value ***m*** which is less than ***n***, then ***m*** (mod ***n***) = ***m***. The original message **M** is generated by applying the reverse of the *padding protocol* to the value ***m***.

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 $2^{16}+1 = 65537$.

What is "computationally expensive" is a relative term depending on when you make the call. 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 increase in size. That holds right up to the point when quantum computing enters the fray.

"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)

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, DNSSEC.

Using RSA in DNSSEC

These days 2,048-bit RSA keys are considered to be a decent size for good security. But if a 2,048-bit key is good, wouldn't a 3,072-bit key be even better? Why don't we all just use massive keys in RSA all of the time? One constraining factor is the cost to generate the keys. The larger the key size the higher the computational load to generate the keys and subsequently generate the encrypted payload. At some point increasing the key size increases the burden placed on the encryptor without substantially altering the security properties of the result. Also, in some areas of application there are very real size constraints placed on the key, 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 where there are some practical constraints on key size in DNSSEC.

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) (RFC 6891), 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. 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 capabilities of the network are far more mundane. IPv4 fragments work reasonably well, but by no means universally. The story in IPv6 is far more dismal than that, and fragmented IPv6 datagrams appear to be discarded in many places within the IPv6 Internet. If you want to reliably receive large blocks of information across any network path, where "large" is any payload greater than 1,240 octets, over either IPv4 of IPv6, then UDP datagrams are definitely not the answer! If you are willing to tolerate some loss, then a payload size of 1,440 octets is the next boundary point, as larger payloads 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 to ensure that the RSA keys cannot be readily cracked using current computational capabilities. Larger keys with DNSSEC implies larger DNS payloads. Larger DNS payloads inevitably pass across the 1,500-octet threshold and IP fragmentation is invoked if we want to continue to use UDP as the transport protocol for DNS queries and responses. But IP packet fragmentation incurs a raised risk of packet loss along the network path. If we want DNSSEC to operate as efficiently as possible with larger RSA keys then we need to avoid this potential packet loss issue, which 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. However, this is a proposition not without its serious 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 DNSSEC-signed responses and operate within the same cost and service parameters if we have to use TCP for all these DNS transactions.

We appear to be wedged between a rock and a hard place here. Today we need to use RSA with 2,048-bit keys in order to ensure an adequately robust cryptographic profile, but that has a consequence in increasing the size of certain DNSSEC responses, which, in turn, may have a much higher loss probability in the network when the increased response size causes the IP datagram to be fragmented.

Thankfully, 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 elliptical curve digital signature algorithm, ECDSA, has something to offer.

ECDSA Cryptography

ECDSA is a digital signature algorithm that is based on a form of cryptography termed Elliptical Curve Cryptography (ECC). 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 an *elliptic curve 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 $\boldsymbol{b}^{\boldsymbol{k}} = \boldsymbol{g}$ for an integer $\boldsymbol{k}$ when $\boldsymbol{b}$ and $\boldsymbol{g}$ 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 (outside of potential approaches using quantum computing of course). The size of the elliptic curve determines the difficulty of the problem.

The major attraction of ECDSA is not necessarily in terms of any claims of superior robustness of the algorithm as compared to RSA, 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.

"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."

RFC 6605, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", P. Hoffman, W.C.A. Wijngaards, April 2012

We are probably justified in being 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. As already noted, if UDP fragmentation in the DNS is unpalatable, then TCP for the DNS may not be 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 makes the shorter key sizes in ECDSA an attractive cryptographic algorithm for use in DNSSEC.

Using ECDSA 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 (so far!). However, there is still a lingering perception issue with certain variants of ECC random bit generation, even though that 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 should we now be confident in using ECDSA on the Internet?

The ECDSA Measurement

The question posed here is: what proportion of the Internet's end users use security-aware 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 (signed with RSA), and
- a *negative* URL that uses a deliberately invalidly-signed DNS name (signed with RSA).

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 *negative* test exposes an interesting side effect of DNS name resolution. There is no *DNSSEC signature verification failure* signal in the DNS, and the DNSSEC designers chose to adopt an existing error code to be backward compatible with existing DNS behaviors. The code chosen for DNSSEC validation failure is Response Code 2 (RCODE 2), otherwise known as SERVFAIL. In other DNS scenarios a SERVFAIL response means "the server you have selected is unable to answer you" which client resolvers interpret as a signal to resend the query to another server. Given that the validation failure will happen for all DNSSEC-enabled queries, the client stub resolver should iterate through all configured recursive resolvers while it attempts to resolve the name. If any of the resolvers is not performing DNSSEC validation the client will be able to resolve the name. So only users who exclusively use DNSSEC validating resolvers will fail to resolve this negative DNS name.

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).

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 security-aware DNS resolvers do when they are confronted with a DNSSEC-signed zone whose signing algorithm is one they don't recognize? RFC 4035 provides the 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.

An 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

We have been measuring the level of support for the ECDSA P-256 as part of the larger APNIC Labs measurement framework for some two years now. The plot of the comparison of users who can validate DNSSEC using RSA signatures and ECDSA P-256 signatures is shown in Figure 1.

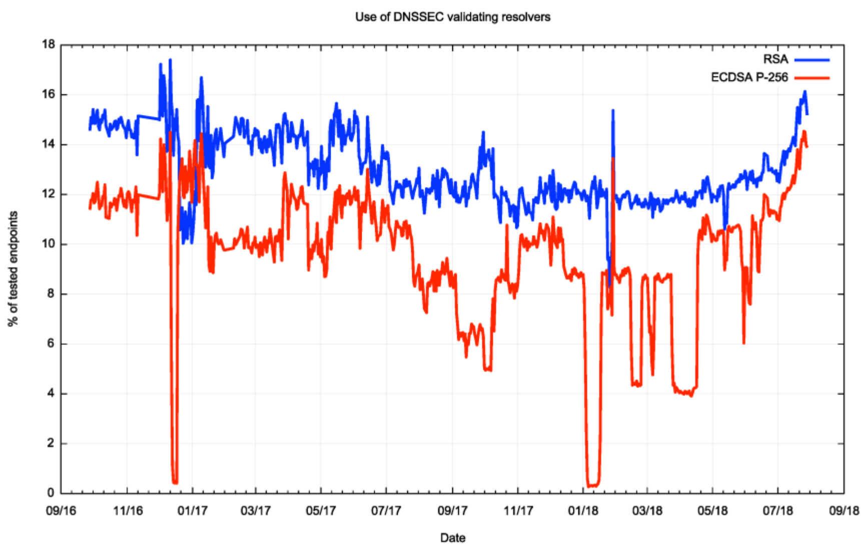

Figure 1 – Support for ECDSA and RSA in Validating DNS resolvers

According to these measurements some 15% of users will perform DNSSEC validation when the DNSSEC signatures use RSA, but the number drops by some 2% when looking at ECDSA signatures. However, while the blue data series (RSA) is consistent across the two year measurement period, the ECDSA measurements (red) show some marked discontinuities. This is due to signature generation failure in the experiment where at times one or more of the servers was serving an invalid ECDSA signature for the validly signed test DNS name. In other words, the sudden drops are due to failures in the measurement setup, and do not reflect changes in the Internet itself.

We still have sufficient data to work from, and we can look at the ratio of ECDSA support to RSA support in DNS resolvers (Figure 2). When we started this measurement in the third quarter of 2016 some 80% of users who exclusively used DNSSEC-validating resolvers were able to correctly validate ECDSA P-256 generated digital signatures. By mid-2018 this number has risen to 90%.

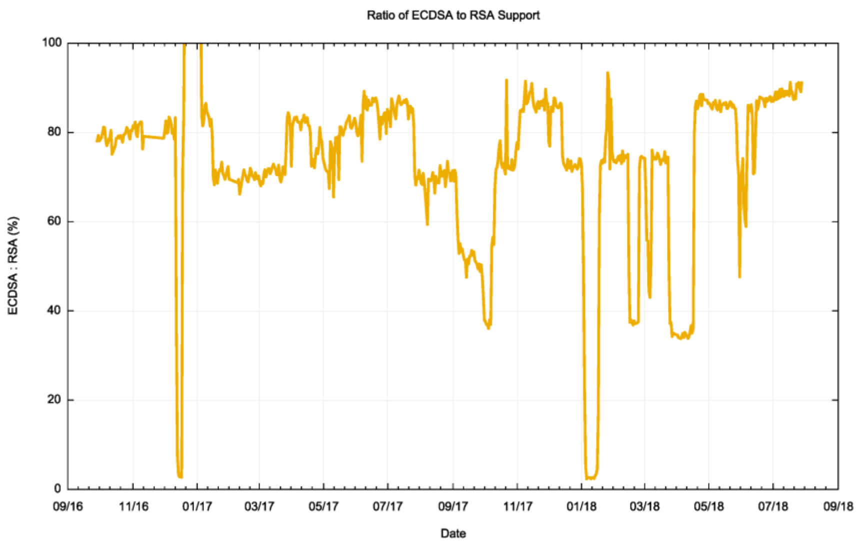

Figure 2 – Ratio of ECDSA to RSA support by users using DNSSEC-validating resolvers

If we discard the periods when one of more of the measurement servers was serving invalid ECDSA signatures and just look at the periods when all four servers were functioning correctly then we can apply a curve fit to the data, as shown in Figure 3.

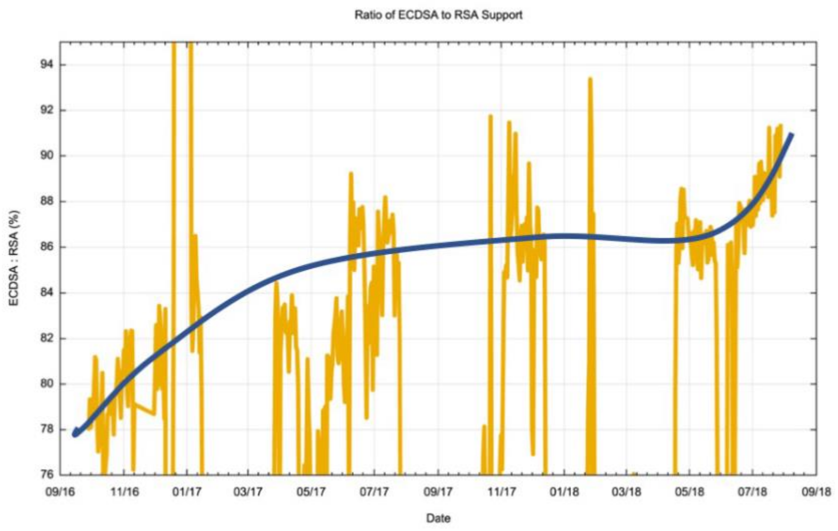

Figure 3 – Ratio of ECDSA to RSA support by users using DNSSEC-validating resolvers

It is evident from this data that there was a visible lift in the level of support for ECDSA in late 2016, and another period of increased level of support in mid 2018. Currently, in more than 90% of cases if a user passes DNS queries to a resolver that performs DNSSEC validation of an RSA digital signature the same resolver will also perform DNSSEC validation of ECDSA P-256 digital signatures. Overall, this is a very encouraging observation.

As with many other technologies, the deployment of support for the ECDSA algorithm is not uniform. While in most areas we have observed security-aware DNS resolution services supporting both RSA and ECDSA, other areas show that ECDSA support is missing. As an example, let's look at data collected from South African users (https://stats.labs.apnic.net/ecdsa/ZA). Figure 4 shows that some 45% of Internet users in South Africa pass their DNS queries to DNS resolvers that perform DNSSEC validation using RSA, yet only one half of those queries reach DNS resolvers that also support validation using ECDSA.

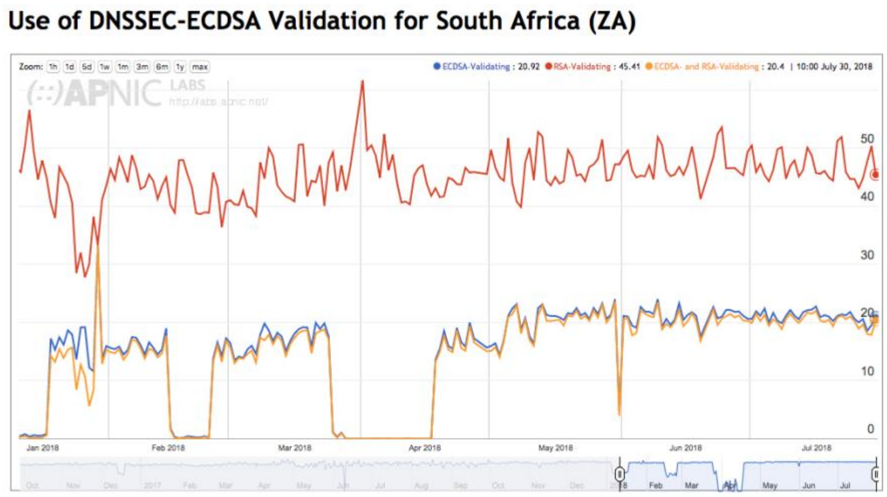

Figure 4 – ECDSA support for South African users

The collected data suggests that one of the larger ISPs in the country, Telkom Internet (AS37457) uses a DNS resolver that does not include ECDSA support. Of the 8 largest ISPs in South Africa, three of these providers appear to be using DNS resolvers with DNSSEC validation enabled. However only one of these three has included support for ECDSA in their resolver configurations.

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.

The map in Figure 5 highlights those countries where there are service providers running DNS resolvers that perform DNSSEC validation, yet do not support ECDSA P-256 crypto.

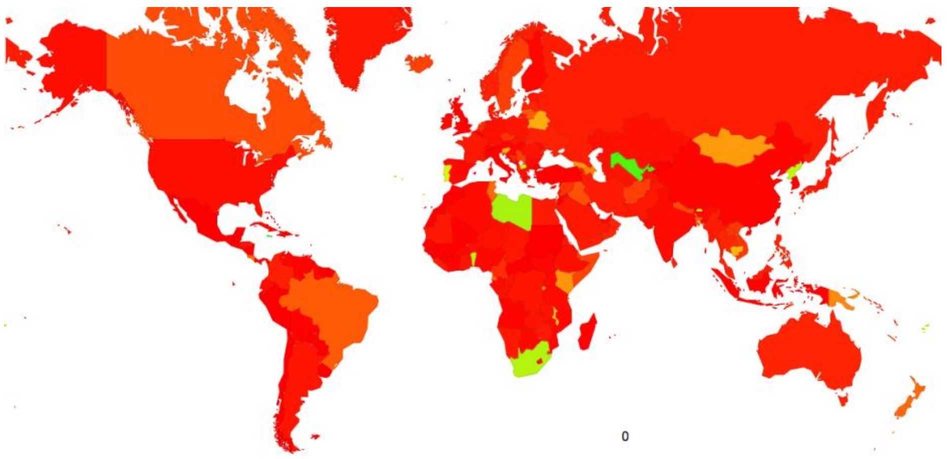

Figure 5 – World Map of DNSSEC Validation not using ECDSA

The economies that are listed here include Uzbekistan (AS8193, BRM and AS41202, UNITEL), Tunisia (AS5438, ATI), Lybia (AS328200, Al-Madar-Al-Jadeed), and South Africa (AS37457, Telkom Internet). It is highly likely that these are instances of recursive DNS resolvers using older distributions of the OpenSSL library that does not include support for the ECDSA P-256 algorithm.

Across the period from the start of June 2018 to mid-August 2018 the measurement experiment saw some 820,000 unique DNS resolvers, as determined by unique IP addresses, ask queries from authoritative name servers. Some 130,000 of these resolvers asked DNSSEC validation queries when the DNA name was signed using the RSA algorithm. However counting resolvers and counting users are somewhat different concepts, and some 35% of users send queries to resolvers that perform DNSSEC validation using RSA crypto.

While some 130,000 resolver IP addresses were seen to perform DNSSEC validation with RSA, only 105,000 resolver IP addresses performed ECDSA P-256 validation. Some 25,000 resolver IP addresses treated the ECDSA P-256 signed DNS names as unsigned due to a lack of local support for this crypto algorithm. However, most of these resolvers are used infrequently, and some 32% of users send queries to resolvers that perform DNSSEC validation using ECDSA P-256 crypto.

What can we say about using ECDSA P-256 as a crypto algorithm for DNSSEC?

On the positive side the digital signatures are far smaller, and DNS UDP responses will be far smaller than comparable responses using RSA. As a general rule in DNS smaller is better and keeping all responses well under the packet fragmentation size is a definite positive. ECDSA P-256 is a compact crypto algorithm and it certainly assists in keeping packet sizes smaller.

On the negative side a small proportion of users, roughly some 3% of the total user population, will fail to validate DNSSEC-signed responses when ECDSA P-256 is used.

However, on the positive side, if a resolver does not support the ECDSA P-256 crypto algorithm the resolver will act as if the DNS data was unsigned. Where there is failure to recognize the signing algorithm, the failure will err on the side of delivering the DNS response.

Another positive aspect is that the total pool of DNS resolvers that fail to support ECDSA P-256 are likely to fall in number as resolvers that run older software configurations are retired or upgraded.

The issue with the poor handling of packet fragmentation in the public Internet is a significant issue and one that resists most conventional efforts to rectify the problem. A prudent server of DNS data would attempt to minimize the incidence of fragmentation-based packet drop. Given that DNSSEC is the major reason why DNS responses can be inflated in size, and larger key sizes in RSA lie behind this packet inflation, changing the crypto algorithm to ECDSA P-256 sounds like a sensible pragmatic decision these days.

Is ECDSA P-256 ready for use?

In my view, this data is now telling us “Yes!”

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.

www.potaroo.net