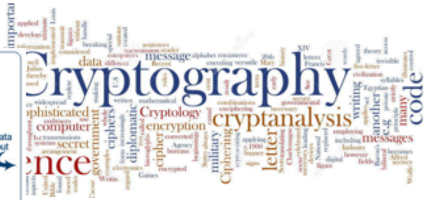
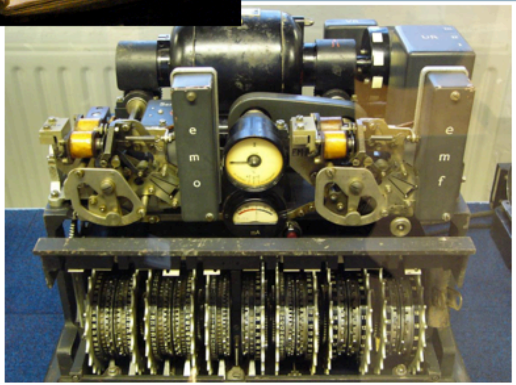
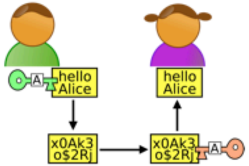
RSA and ECDSA

Geoff Huston
APNIC

It's all about Cryptography







Why use Cryptography?

Public key cryptography can be used in a number of ways:

- protecting a session from third party eavesdroppers

  Encryption using a session key that is known only to the parties to the conversation

- protecting a session from interference

  Injection (or removal) of part of a session can only be undertaken by the parties to the session

- authentication and non-repudiation

  What is received is exactly what the other party sent, and cannot be repudiated

Symmetric Crypto

A symmetric crypto algorithm uses the same key to

- Convert a plaintext message to a crypted message
- Convert a crypted message to its plaintext message

- They are generally fast and simple

BUT they use a shared key

- This key distribution problem can be a critical weakness in the crypto framework

Asymmetric Crypto

This is a class of asymmetric transforms applied to a message such that:

Messages encrypted using Key A and algorithm X can only be translated back to the original message using Key B and algorithm X

This also holds in reverse

This can address the shared key problem:

If I publish Key A and keep Key B a secret then you can send me a secret by encrypting it using my public key A

The Asymmetric Crypto Challenge

Devise an algorithm (encoding) and keys such that:

- Messages encoded with one key can only be decoded with the other key
- Knowledge of the value of one key does not infer the value of the other key

http://bit.ly/2iQ0oi7

RSA

Select two large (> 256 bit) prime numbers, *p* and *q, then:*

$n = p.q$

$\phi(n) = (p-1).(q-1)$ *(the number of numbers that are relatively prime to n)*

Pick an *e* that is relatively prime to $\phi(n)$

The PUBLIC KEY is <e,n>

Pick a value for *d* such that $d.e = 1 \bmod \phi(n)$

The PRIVATE KEY is <d,n>

For any x, $x^{de} \equiv x \pmod{n}$

Why does RSA work?

Encryption using the public key consists of taking a message *x* and raising it to the power *e*

$$Crypt = x^e$$

Decryption consists of taking an encrypted message and raising it to the power d, mod n

$$Decrypt = Crypt^d \bmod n = (x^e)^d \bmod n = x^{ed} \bmod n = x$$

Similarly, one can encrypt a message with the private key (x^d) and decrypt with the public key ($(x^d)^e \bmod n = x$)

Why does RSA work?

If you know e and n (the public key) then how can you calculate d (the private key)?

Now $d.e = 1 \bmod \phi(n)$

If you know $\phi(n)$ you can calculate d

But $\phi(n) = (p-1).(q-1)$, where $p.q = n$

i.e. you need to find the prime factors of n, a large composite number that is the product of two primes

The 'core' of RSA

$$(x^e)^d \equiv x \mod n$$

As long as *d* and *n* are relatively large, and *n* is the product of two large prime numbers, then finding the value of *d* when you already know the values of *e* and *n* is computationally expensive

The 'core' of RSA

$$(x^e)^d \equiv x \mod n$$

As long as *d* and *n* are relatively large, and *n* is the product of two large prime numbers, then finding the value of *d* when you already know the values of *e* and *n* is computationally expensive

But computers get larger and faster – what was infeasible yesterday may be possible tomorrow

The 'core' of RSA

$$(x^e)^d \equiv x \mod n$$

As long as *d* and *n* are relatively large, and *n* is the
product of two large prime numbers, then finding the value of *d* when you already know the values of *e* and *n* is computationally expensive

But computers get larger and faster – what was infeasible yesterday may be possible tomorrow

The way to stay ahead is to make the value of n larger and larger

Why is this important?

Because much of the foundation of internet Security rests upon this relationship

How big can RSA go?

In theory we can push this to very large sizes of *n* to generate RSA private keys

The algorithm is not itself arbitrarily limited in terms of key size

But as the numbers get larger there is higher computation overhead to generate and manipulate these keys

So we want it large enough not to be ‘broken’ by most forms of brute force, but small enough to be computed by our everyday processors

How big should RSA go?

You need to consider time as well

How long do you want or need your secret to remain a secret?

- Because if the attacker has enough time a brute force attack may work

- Also time is on the attacker's side: keys that are considered robust today may not be as robust tomorrow, assuming that feasible compute capabilities rise over time

So you want to pick a key size that is resistant to attempts to brute force the key both today and tomorrow

Bigger and bigger?

Well, no – the larger the key sizes compared to compute capabilities means:

- Longer times to generate keys
- Longer times to encrypt (and decrypt) messages
- More space to represent the key values

So you need to use big keys, but no bigger then necessary!

Be Specific!

Time to consult the experts!

http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

NIST Special Publication 800-57 Part 3
Revision 1

Recommendation for Key Management

Part 3: Application-Specific Key Management Guidance

Elaine Barker
Quynh Dang

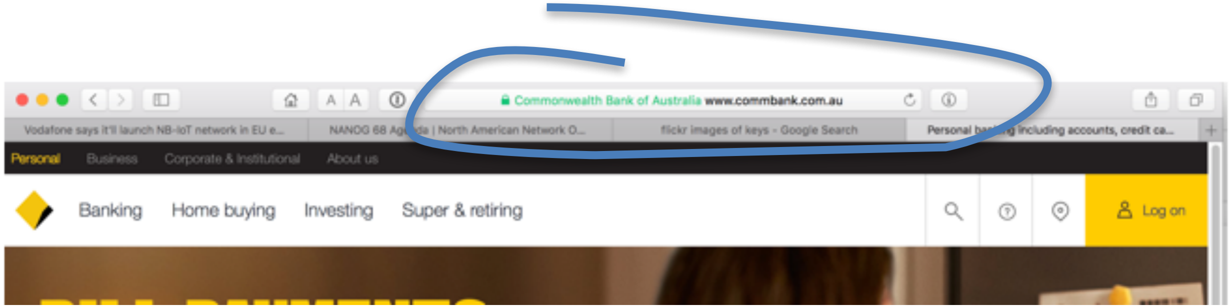
This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-57pt3r1

Table 2-1: Recommended Algorithms and Key Sizes

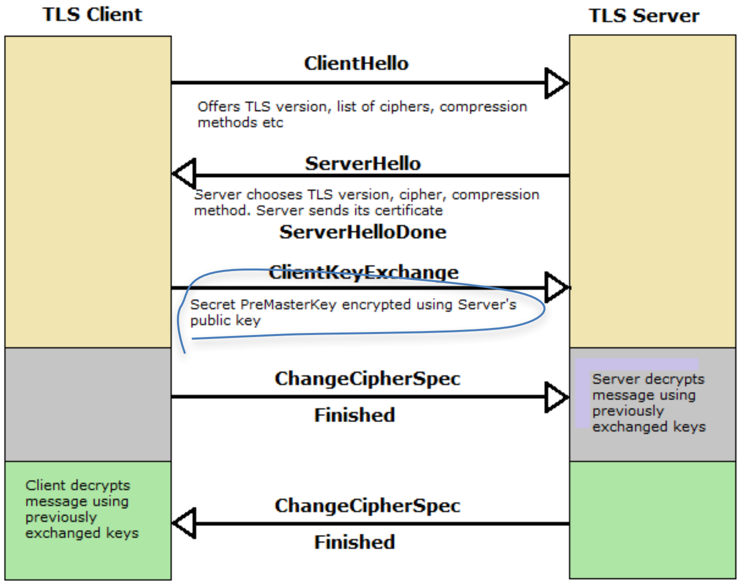
Key Type	Algorithms and Key Sizes
Digital Signature keys used for authentication (for Users or Devices)	RSA (2048 bits) ECDSA (Curve P-256)
Digital Signature keys used for non-repudiation (for Users or Devices)	RSA (2048 bits) ECDSA (Curves P-256 or P-384)
CA and OCSP Responder Signing Keys	RSA (2048 or 3072bits) ECDSA (Curves P-256 or P-384)
Key Establishment keys (for Users or Devices)	RSA (2048 bits) Diffie-Hellman (2048 bits) ECDH (Curves P-256 or P-384)

RSA is everywhere…

My Bank…(I hope!)

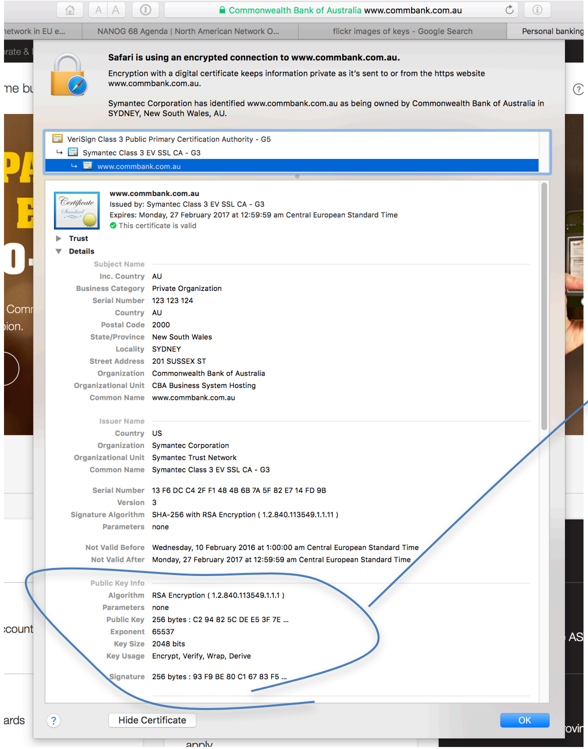


TLS: Protecting the session



https://rhsecurity.wordpress.com/tag/tls/

The Key to My Bank



Yes, the fine print says my bank is using a 2048-bit RSA Public key to as the foundation of the session key used to secure access to my bank

I trust its my bank because …

- The server has demonstrated knowledge of a private key that is associated with a public key that I have been provided
- The public key has been associated with a particular domain name by a Certificate Authority
- My browser trusts that this Certificate Authority never lies about such associations
- So if the server can demonstrate that it has the private key then my browser will believe that its my bank!

DNSSEC and the DNS

Another major application for crypto in the Internet is securing the DNS

You want to be assured that the response you get to from DNS query is:

- Authentic
- Complete
- Current

DNSSEC Interlocking Signatures

. (root)

. Key-Signing Key – signs over

. Zone-Signing Key – signs over

DS for .com (Key-Signing Key)

.com

.com Key-Signing Key – signs over

.com Zone-Signing Key – signs over

DS for example .com (Key-Signing Key)

.example.com

example.com Key-Signing Key – signs over

example.com Zone-Signing Key – signs over

www.example.com

www.example.com

DNSSEC Interlocking Signatures

. (root)

. Key-Signing Key – signs over

. Zone-Signing Key – signs over

DS for .com (Key-Signing Key)

.com

.com Key-Signing Key – signs over

.com Zone-Signing Key – signs over

DS for example .com (Key-Signing Key)

.example.com

example.com Key-Signing Key – signs over

example.com Zone-Signing Key – signs over

www.example.com

www.example.com IN A 192.0.1

DNSSEC Interlocking Signatures

. (root)

. Key-Signing Key – signs over

. Zone-Signing Key – signs over

DS for .com (Key-Signing Key)

.com

.com Key-Signing Key – signs over

.com Zone-Signing Key – signs over

DS for example .com (Key-Signing Key)

.example.com

example.com Key-Signing Key – signs over

example.com Zone-Signing Key – signs over

www.example.com

www.example.com IN A 192.0.1

is the KSK for . valid?

is the ZSK for . valid?

is this DS equal to the hash of the KSK?
is the signature for this record valid?

is the KSK for .com valid?

is the ZSK for .com valid?

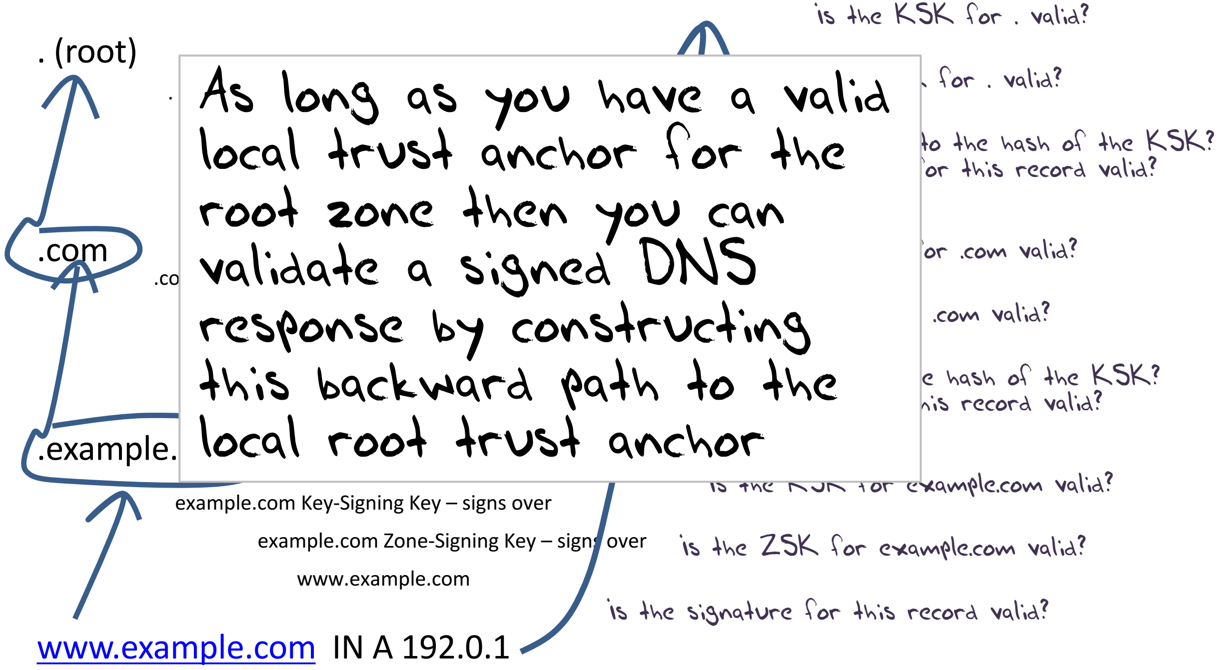
is this DS equal to the hash of the KSK?
is the signature for this record valid?

is the KSK for example.com valid?

is the ZSK for example.com valid?

is the signature for this record valid?

DNSSEC Interlocking Signatures



As long as you have a valid local trust anchor for the root zone then you can validate a signed DNS response by constructing this backward path to the local root trust anchor

A DNSSEC response using RSA

```
$ dig +dnssec u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net

; <<>> DiG 9.9.6-P1 <<>> +dnssec u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25461
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net. IN A

;; ANSWER SECTION:
u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net. 1      IN A 199.102.79.186
u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net. 1      IN RRSIG A 5 4 3600 20200724235900 20130729104013 1968 5a593.z.dotnxdomain.net. ghHPoQd71aZtsdH823eW

;; AUTHORITY SECTION:
33d23a33.3b7acf35.9bd5b553.3ad4aa35.09207c36.a095a7ae.1dc33700.103ad556.3a564678.16395067.a12ec545.6183d935.c68cebfb.41a4008e.4f291b87.479c6f9e.5ea48f86.7d1187f1.7572d59a
33d23a33.3b7acf35.9bd5b553.3ad4aa35.09207c36.a095a7ae.1dc33700.103ad556.3a564678.16395067.a12ec545.6183d935.c68cebfb.41a4008e.4f291b87.479c6f9e.5ea48f86.7d1187f1.7572d59a
5a593.z.dotnxdomain.net. 3599 IN        NS      nsz1.z.dotnxdomain.net.
5a593.z.dotnxdomain.net. 3600 IN        RRSIG   NS 5 4 3600 20200724235900 20130729104013 1968 5a593.z.dotnxdomain.net. ntxWo5UwL1vQjOHY0z5DCVNDDScnd3Tglgd0PsBRRhk3B9iJ

;; Query time: 1052 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Mar 12 03:59:57 UTC 2015
;; MSG SIZE  rcvd: 937
```

RSA signed response – 937 octets

Another DNSSEC response using RSA

```
$ dig +dnssec DNSKEY org

; <<>> DiG 9.11.0-P1 <<>> +dnssec DNSKEY org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53713
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 1

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

;; ANSWER SECTION:
org.                    900     IN      DNSKEY  256 3 7 AwEAAXxsMmN/JgpEE9Y4uFNRJm7Q9GBwmEYUCsCxuKlgBU9WrQEFRrvA eMamUBeX4SE
org.                    900     IN      DNSKEY  256 3 7 AwEAAayiVbuM+ehlsKsuAL1CI3mA+5JM7ti3VeY8ysmogElVMuSLNsX7 HFyq9O6qhZV
org.                    900     IN      DNSKEY  257 3 7 AwEAAcMnWBKLuvG/LwnPVykcmpvnntwxfshHlHRhlY0F3oz8AMcuF8gw 9McCw+BoC2Y
org.                    900     IN      DNSKEY  257 3 7 AwEAAZTjbIO5kIpxWUtyXc8avsKyHIIZ+LjC2Dv8naO+Tz6X2fqzDC1b dq7HlZwtkaq
org.                    900     IN      RRSIG   DNSKEY 7 1 900 20170207153219 20170117143219 3947 org. S6+vpFWz6hfPmvI7zxRa4
org.                    900     IN      RRSIG   DNSKEY 7 1 900 20170207153219 20170117143219 9795 org. iEyiroy02ljtH5hf5RIdf
org.                    900     IN      RRSIG   DNSKEY 7 1 900 20170207153219 20170117143219 17883 org. A2hLUswcas+W4h8gZYpA

;; Query time: 475 msec
;; SERVER: 203.133.248.1#53(203.133.248.1)
;; WHEN: Thu Jan 19 23:37:38 UTC 2017
;; MSG SIZE  rcvd: 1625
```

RSA signed response – 1,625 octets

Not every application can tolerate large keys…

The DNS and DNSSEC is a problem here:

- including the digital signature increases the response size
- Large responses generate packet fragmentation
- Fragments are commonly filtered by firewalls
- IPv6 Fragments required IPv6 Extension Headers, and packets with Extension Headers are commonly filtered
- DNS over TCP imposes server load
- DNS over TCP is commonly filtered

If you **can** avoid large responses in the DNS, you **should**!

The search for small keys

- Large keys and the DNS don’t mix very well:
  - We try and make UDP fragmentation work reliably (for once!)
  - Or we switch the DNS to use TCP
  - Or we look for smaller keys

Enter Elliptic Curves

Alice creates a key pair, consisting of a private key integer d_A, randomly selected in the interval $[1, n-1]$; and a public key curve point $Q_A = d_A \times G$. We use $\times$ to denote elliptic curve point multiplication by a scalar.

For Alice to sign a message m, she follows these steps:

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \mod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + r d_A) \mod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

When computing s, the string z resulting from $\mathrm{HASH}(m)$ shall be converted to an integer. Note that z can be *greater* than n but not *longer*.[1]

As the standard notes, it is crucial to select different k for different signatures, otherwise the equation in step 6 can be solved for d_A, the private key: Given two signatures (r, s) and (r, s'), employing the same unknown k for different known messages m and m', an attacker can calculate z and z', and since $s - s' = k^{-1}(z - z')$ (all operations in this paragraph are done modulo n) the attacker can find $k = \frac{z - z'}{s - s'}$. Since $s = k^{-1}(z + r d_A)$, the attacker can now calculate the private key $d_A = \frac{sk - z}{r}$. This implementation failure was used, for example, to extract the signing key used in the PlayStation 3 gaming-console.[2] Another way ECDSA signature may leak private keys is when k is generated by a faulty random number generator. Such a failure in random number generation caused users of Android Bitcoin Wallet to lose their funds in August 2013.[3] To ensure that k is unique for each message one may bypass random number generation completely and generate deterministic signatures by deriving k from both the message and the private key.[4]

Signature verification algorithm [edit]

For Bob to authenticate Alice's signature, he must have a copy of her public-key curve point Q_A. Bob can verify Q_A is a valid curve point as follows:

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid
2. Check that Q_A lies on the curve
3. Check that $n \times Q_A = O$

After that, Bob follows these steps:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \mod n$.
5. Calculate $u_1 = zw \mod n$ and $u_2 = rw \mod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

Note that using Shamir's trick, a sum of two scalar multiplications $u_1 \times G + u_2 \times Q_A$ can be calculated faster than two scalar multiplications done independently.[5]

Correctness of the algorithm [edit]

It is not immediately obvious why verification even functions correctly. To see why, denote as C the curve point computed in step 6 of verification,

$$C = u_1 \times G + u_2 \times Q_A$$

From the definition of the public key as $Q_A = d_A \times G$,

$$C = u_1 \times G + u_2 d_A \times G$$

Because elliptic curve scalar multiplication distributes over addition,

$$C = (u_1 + u_2 d_A) \times G$$

Expanding the definition of u_1 and u_2 from verification step 5,

$$C = (zs^{-1} + r d_A s^{-1}) \times G$$

Collecting the common term s^{-1},

$$C = (z + r d_A) s^{-1} \times G$$

Expanding the definition of s from signature step 6,

$$C = (z + r d_A)(z + r d_A)^{-1}(k^{-1})^{-1} \times G$$

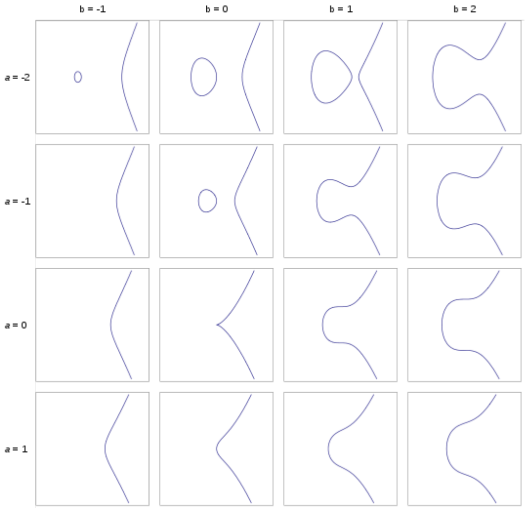
Since the inverse of an inverse is the original element, and the product of an element's inverse and the element is the identity, we are left with

$$C = k \times G$$

From the definition of r, this is verification step 6.

This shows only that a correctly signed message will verify correctly; many other properties are required for a secure signature algorithm.

$$y^2 = x^3 + ax + b$$



Enter Elliptic Curves

Alice creates a key pair, consisting of a private key integer d_A, randomly selected in the interval $[1, n-1]$; and a public key curve point $Q_A = d_A \times G$. We use $\times$ to denote elliptic curve point multiplication by a scalar.

For Alice to sign a message m, she follows these steps:

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \bmod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \bmod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

"It is not immediately obvious why verification even functions correctly." !!

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid
2. Check that Q_A lies on the curve
3. Check that $n \times Q_A = O$

After that, Bob follows these steps:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \bmod n$.
5. Calculate $u_1 = zw \bmod n$ and $u_2 = rw \bmod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

Note that using Shamir's trick, a sum of two scalar multiplications $u_1 \times G + u_2 \times Q_A$ can be calculated faster than two scalar multiplications done independently.[5]

Correctness of the algorithm [edit]

It is not immediately obvious why verification even functions correctly. To see why, denote as C the curve point computed in step 6 of verification,

$$C = u_1 \times G + u_2 \times Q_A$$

From the definition of the public key as $Q_A = d_A \times G$,

$$C = u_1 \times G + u_2 d_A \times G$$

Because elliptic curve scalar multiplication distributes over addition,

$$C = (u_1 + u_2 d_A) \times G$$

Expanding the definition of u_1 and u_2 from verification step 5,

$$C = (zs^{-1} + rd_A s^{-1}) \times G$$

Collecting the common term s^{-1},

$$C = (z + rd_A)s^{-1} \times G$$

Expanding the definition of s from signature step 6,

$$C = (z + rd_A)(z + rd_A)^{-1}(k^{-1})^{-1} \times G$$

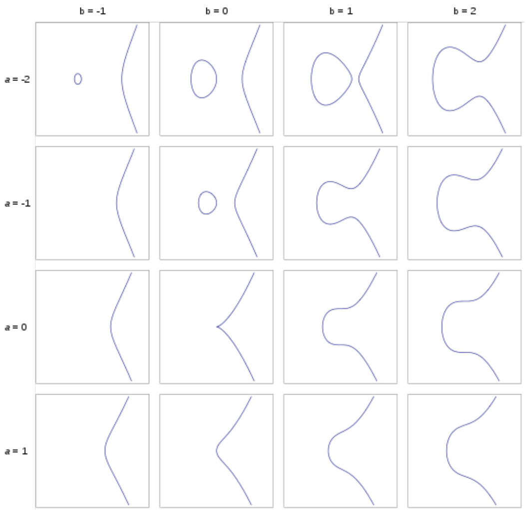
Since the inverse of an inverse is the original element, and the product of an element's inverse and the element is the identity, we are left with

$$C = k \times G$$

From the definition of r, this is verification step 6.

This shows only that a correctly signed message will verify correctly; many other properties are required for a secure signature algorithm.

$$y^2 = x^3 + ax + b$$



ECDSA P-256

Elliptic Curve Cryptography allows for the construction of “strong” public/private key pairs with key lengths that are far shorter than equivalent strength keys using RSA

A 256-bit ECC key should provide comparable security to a 3072-bit RSA key

ECDSA vs RSS

```
$ dig +dnssec u5221730329.s1425859199.i5075.vcf100.5a593.y.dotnxdomain.net

; <<>> DiG 9.9.6-P1 <<>> +dnssec u5221730329.s1425859199.i5075.vcf100.5a593.y.dot
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61126
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;u5221730329.s1425859199.i5075.vcf100.5a593.y.dotnxdomain.net. IN A

;; ANSWER SECTION:
u5221730329.s1425859199.i5075.vcf100.5a593.y.dotnxdomain.net. 1      IN A 144.76
u5221730329.s1425859199.i5075.vcf100.5a593.y.dotnxdomain.net. 1      IN RRSIG A 1

;; AUTHORITY SECTION:
ns1.5a593.y.dotnxdomain.net. 1          IN        NSEC      x.5a593.y.dotnxdomain
ns1.5a593.y.dotnxdomain.net. 1          IN        RRSIG     NSEC 13 5 1 202007242
5a593.y.dotnxdomain.net. 3598 IN        NS        ns1.5a593.y.dotnxdomain.net.
5a593.y.dotnxdomain.net. 3600 IN        RRSIG     NS 13 4 3600 20200724235900 201

;; Query time: 1880 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Mar 12 03:59:42 UTC 2015
;; MSG SIZE  rcvd: 527
```

ECDSA signed response – 527 octets

```
$ dig +dnssec u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.ne

; <<>> DiG 9.9.6-P1 <<>> +dnssec u5221730329.s1425859199.i5075.vcf100.5a5        nxdomain.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25461
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net. IN A

;; ANSWER SECTION:
u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net. 1      IN A        2.79.186
u5221730329.s1425859199.i5075.vcf100.5a593.z.dotnxdomain.net. 1      IN R        5 4 3600 2020072423590

;; AUTHORITY SECTION:
33d23a33.3b7acf35.9bd5b553.3ad4aa35.09207c36.a095a7ae.1dc33700.103ad556.3        16395067.a12ec545.618
33d23a33.3b7acf35.9bd5b553.3ad4aa35.09207c36.a095a7ae.1dc33700.103ad556.3        16395067.a12ec545.618
5a593.z.dotnxdomain.net. 3599 IN        NS        nsz1.z.dotnxdomain.net.
5a593.z.dotnxdomain.net. 3600 IN        RRSIG     NS 5 4 3600 2020072423        0729104013 1968 5a593.

;; Query time: 1052 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Mar 12 03:59:57 UTC 2015
;; MSG SIZE  rcvd: 937
```

RSA signed response – 937 octets

ECDSA has a history…

Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book

ECC patents

From Wikipedia, the free encyclopedia

Patent-related uncertainty around elliptic curve cryptography (ECC), or **ECC patents**, is one of the main factors limiting its wide acceptance. For example, the OpenSSL team accepted an ECC patch only in 2005 (in OpenSSL version 0.9.8), despite the fact that it was submitted in 2002.

According to Bruce Schneier as of May 31, 2007, "Certicom certainly can claim ownership of ECC. The algorithm was developed and patented by the company's founders, and the patents are well written and strong. I don't like it, but they can claim ownership."[1] Additionally, NSA has licensed MQV and other ECC patents from Certicom in a US$25 million deal for NSA Suite B algorithms.[2] (ECMQV is no longer part of Suite B.)

However, according to RSA Laboratories, "*in all of these cases, it is the implementation technique that is patented, not the prime or representation, and there are alternative, compatible implementation techniques that are not covered by the patents.*"[3] Additionally, Daniel J. Bernstein has stated that he is "not aware of" patents that cover the Curve25519 elliptic curve Diffie–Hellman algorithm or its implementation.[4] RFC 6090, published in February 2011, documents ECC techniques, some of which were published so long ago that even if they were patented any such patents for these previously published techniques would now be expired.

Contents [hide]

ECDSA and OpenSSL

- OpenSSL added ECDSA support as from 0.9.8 (2005)
- Other bundles and specific builds added ECDSA support later
- But deployed systems often lag behind the latest bundles, and therefore still do not include ECC support in their running configuration

Is ECDSA viable?

What does NIST say?

http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

NIST Special Publication 800-57 Part 3
Revision 1

Recommendation for
Key Management

Part 3: Application-Specific Key Management Guidance

Elaine Barker
Quynh Dang

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-57pt3r1

Table 2-1: Recommended Algorithms and Key Sizes

Key Type	Algorithms and Key Sizes
Digital Signature keys used for authentication (for Users or Devices)	RSA (2048 bits) ECDSA (Curve P-256)
Digital Signature keys used for non-repudiation (for Users or Devices)	RSA (2048 bits) ECDSA (Curves P-256 or P-384)
CA and OCSP Responder Signing Keys	RSA (2048 or 3072bits) ECDSA (Curves P-256 or P-384)
Key Establishment keys (for Users or Devices)	RSA (2048 bits) Diffie-Hellman (2048 bits) ECDH (Curves P-256 or P-384)

Do folk use ECDSA for public keys?

```
$ dig +dnssec www.cloudflare-dnssec-auth.com

; <<>> DiG 9.9.6-P1 <<>> +dnssec www.cloudflare-dnssec-auth.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7049
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.cloudflare-dnssec-auth.com.      IN      A

;; ANSWER SECTION:
www.cloudflare-dnssec-auth.com.       300 IN  A       104.20.23.140
www.cloudflare-dnssec-auth.com.       300 IN  A       104.20.21.140
www.cloudflare-dnssec-auth.com.       300 IN  A       104.20.19.140
www.cloudflare-dnssec-auth.com.       300 IN  A       104.20.22.140
www.cloudflare-dnssec-auth.com.       300 IN  A       104.20.20.140
www.cloudflare-dnssec-auth.com.       300 IN  RRSIG   A 13 3 300 20150317021923 20150315001923 35273
cloudflare-dnssec-auth.com. pgBvfQkU4Il8ted2hGL9o8NspvKksDT8/jvQ+4o4h4tGmAX0fDBEoorb
tLiW7mcdOWYLoOnjovzYh3QOodu0Xw==

;; Query time: 237 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Mon Mar 16 01:19:24 UTC 2015
;; MSG SIZE  rcvd: 261
```

Algorithm 13 is ECDSA P-256

Signed response is 261 octets long!

So lets use ECDSA for DNSSEC

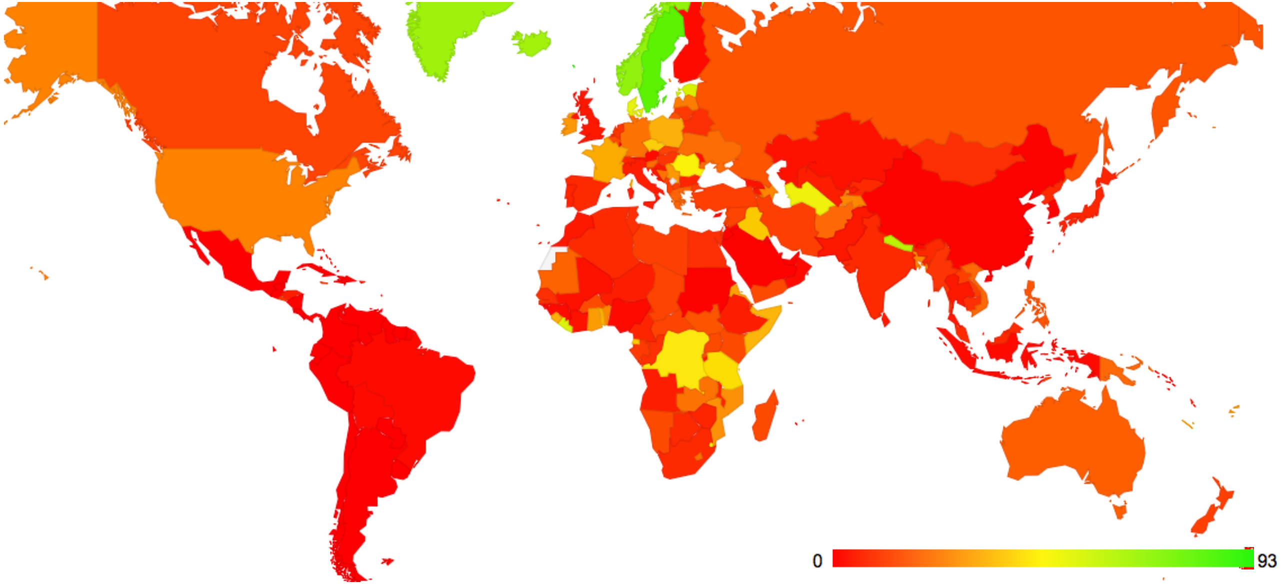
Or maybe we should look before we leap...

– Is ECDSA a "well supported" crypto protocol? *

– If you signed using ECDSA would resolvers validate the signature?

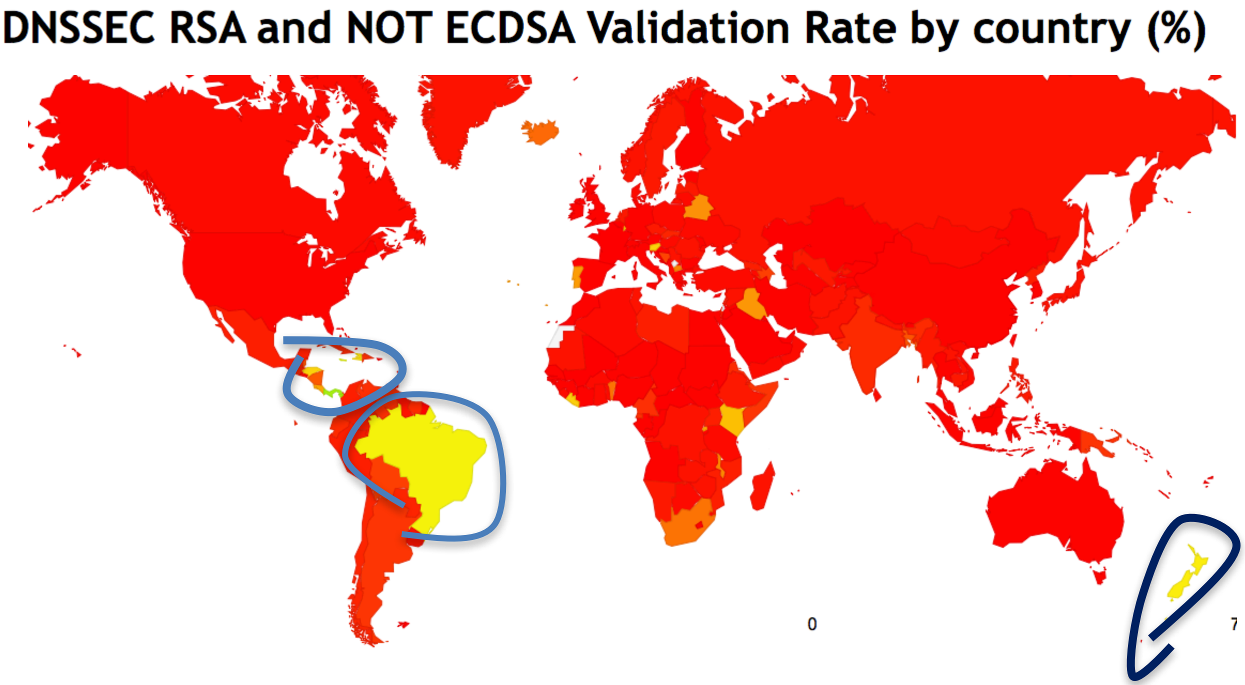
It's not that crypto libraries deliberately exclude ECDSA support these days. The more likely * issue appears to be the operational practic es of some ISPs who use crufty old software sets to support DNS resolvers which are now running old libraries that predate the incorporation of ECDSA into Open SSL

Where are the users who can validate ECDSA-signed DNSSEC records?



https://stats.labs.apnic.net/ecdsa

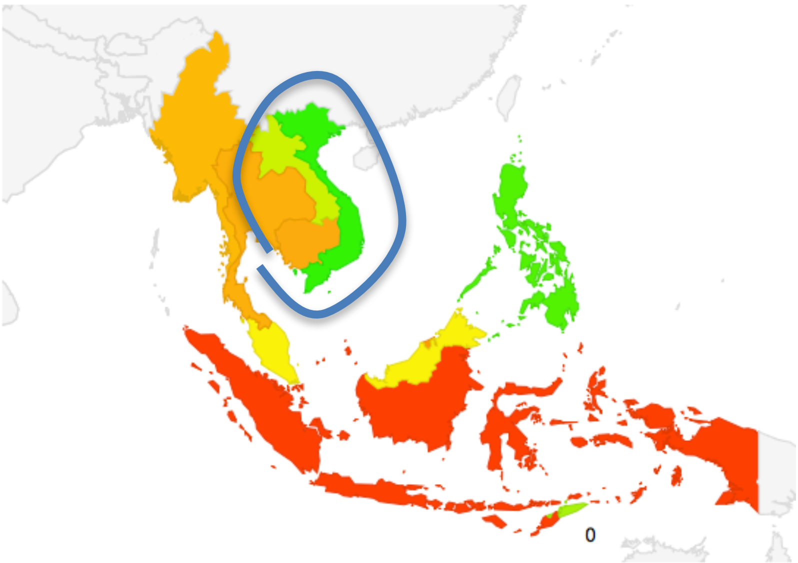
And where ECDSA support is missing



https://stats.labs.apnic.net/ecdsa

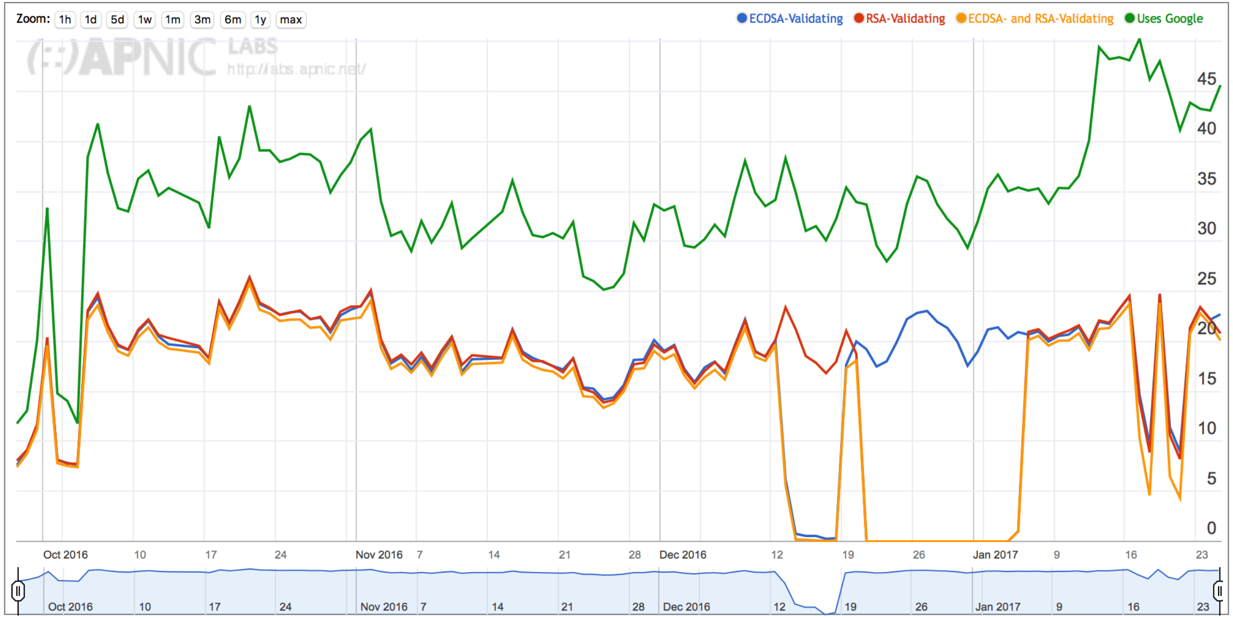
Today we're in Vietnam…

Region Map for South-Eastern Asia (035)



Today we're in Vietnam…

Use of DNSSEC-ECDSA Validation for Vietnam (VN)



The Top 5 Vietnam ISPs

ASN	AS Name	ECDSA Validates	RSA Validates	ECDSA and RSA Validates	ECDSA : RSA Ratio (%)	Uses Google PDNS	Samples
AS45899	VNPT-AS-VN VNPT Corp	22.21%	17.27%	14.88%	100.00%	41.95%	7,749,061
AS7552	VIETEL-AS-AP Viettel Corporation	21.30%	17.71%	15.79%	100.00%	34.51%	4,240,602
AS18403	FPT-AS-AP The Corporation for Financing Promoting Technology	19.75%	17.10%	14.88%	100.00%	31.98%	3,985,424
AS131178	KINGCORP-AS-IX Opennet Internet Exchange	5.28%	4.72%	4.12%	100.00%	13.75%	2,939,888
AS24086	VIETTEL-AS-VN Viettel Corporation	12.52%	10.28%	8.71%	100.00%	23.01%	1,349,824

And the extent to which their uses perform DNSSEC validation with ECDSA and RSA

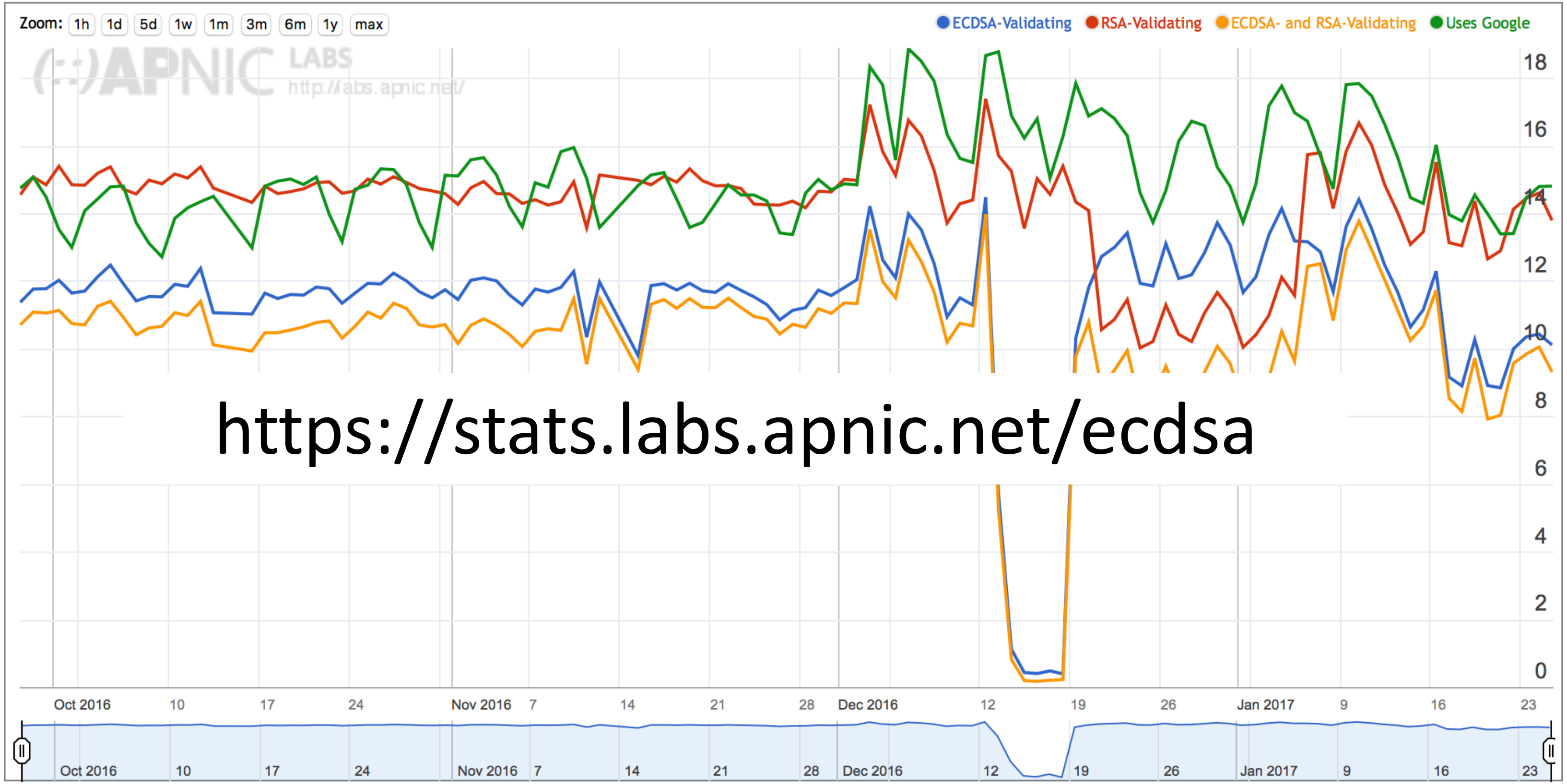
And it if wasn't for Google…

ASN	AS Name	ECDSA Validates	RSA Validates	ECDSA and RSA Validates	ECDSA : RSA Ratio (%)	Uses Google PDNS	Samples
AS45899	VNPT-AS-VN VNPT Corp	22.21%	17.27%	14.88%	100.00%	41.95%	7,749,061
AS7552	VIETEL-AS-AP Viettel Corporation	21.30%	17.71%	15.79%	100.00%	34.51%	4,240,602
AS18403	FPT-AS-AP The Corporation for Financing Promoting Technology	19.75%	17.10%	14.88%	100.00%	31.98%	3,985,424
AS131178	KINGCORP-AS-IX Opennet Internet Exchange	5.28%	4.72%	4.12%	100.00%	13.75%	2,939,888
AS24086	VIETTEL-AS-VN Viettel Corporation	12.52%	10.28%	8.71%	100.00%	23.01%	1,349,824

There would probably be no DNSSEC at all!

And no ECDSA!

APNIC Labs Report on ECDSA use



https://stats.labs.apnic.net/ecdsa

Thanks!

Me: gih@apnic.net