Quest for missing keytags

Roy Arends | DNS-OARC | 1 April 2016

Pubquiz question:

What is a DNSKEY Key Tag

A. a 16 bit value in the DNSKEY RDATA

B. a physical tag that you’d hang on your key ring

C. a 16 bit value in the DS and RRSIG RDATA

D. a special variation of the game of tag.

Why did I look into this?

2010, first root KSK published,
2015, I started working on my testbed

Why did I look into this?

2010
2015

Why did I look into this?

2010
2015

I wanted to use those as keytags for my testbed.

You can’t simply assign a keytag to a dnskey.

RFC4034:

“the algorithm for calculating the Key Tag is almost but not completely identical to the familiar ones-complement checksum used in many other Internet protocols.”

Simple loop

```
while true
   do dnssec-keygen -a RSASHA256 -f KSK -b 2048 .
done
```

Simple loop

```
while true
   do dnssec-keygen -a RSASHA256 -f KSK -b 2048 .
done
```

This only generated about 16K keys

I was expecting 64K keys

keytags 02010 and 02015 were absent

Simple loop

```
while true
   do dnssec-keygen -a RSASHA256 -f KSK -b 2048 .
done
```

First clue by Duane Wessels:

dnssec-keygen won't generate a new key if:

- the new key tag conflicts with an existing key tag + revoke bit
- the new key tag + revoke bit conflicts with an existing key tag

Nice! Well observed.

Less Simple loop

```
while true
   do dnssec-keygen -a RSASHA256 -f KSK -b 2048 .\
   >> taglist
   rm K\.+008*;
done
```

This simply removes keys after they're created, but adds the tag to a list.

Less Simple loop

```
while true
   do dnssec-keygen -a RSASHA256 -f KSK -b 2048 .\
   >> taglist
   rm K\.+008*;
done
```

This simply removes keys after they're created, but adds the tag to a list.

```
sort -u taglist | wc -l
16387
```

Less Simple loop

```
while true
   do dnssec-keygen -a RSASHA256 -f KSK -b 2048 .\
   >> taglist
   rm K\.+008*;
done
```

This simply removes keys after they're created, but adds the tag to a list.

```
sort -u taglist | wc -l
16387
```

Wait, what? Not 16384?

It’s the tool, try a different one.

```
while true
   do ldns-keygen -a RSASHA256 -k -b 2048 .
done
```

Nice and simple. No undocumented features.

Allows for foot shooting.

It’s the tool, try a different one.

```
while true
   do ldns-keygen -a RSASHA256 -k -b 2048 .
done
```

Nice and simple. No undocumented features.

```
ls K.*private | wc -l
16385
```

It’s the tool, try a different one.

```
while true
   do ldns-keygen -a RSASHA256 -k -b 2048 .
done
```

Nice and simple. No undocumented features.

```
ls K.*private | wc -l
16385
```

Still, not high enough.

Meanwhile, via Twitter

Peter van Dijk
@Habbie

Following

@royarends just as a data point I now have a collection of 2048bit RSASHA256 keys with 17896 distinct keytags. Still generating more keys :)

LIKES
2


1:14 PM - 30 Nov 2015

Meanwhile, via Twitter

Peter van Dijk
@Habbie

Following

@royarends the tool is 'pdnssec add-zone-key' using mbedTLS 2.1.0 (formerly known as Polar). Flags all 257. I'll check the exponents.

1:45 PM - 30 Nov 2015

So, it could be the library

DNSSEC-Keygen and ldns-keygen use OpenSSL

pdnssec uses mbedTLS

Is this a bug in OpenSSL?

So, it could be the library

DNSSEC-Keygen and ldns-keygen use OpenSSL

pdnssec uses mbedTLS

Is this a bug in OpenSSL?

“KEYSTARVE” [goes and registers name]

But…. On DNS-Operations

Peter van Dijk:

I now have ~130k (different!) keys, with 32201 unique key tags. This is almost twice as much as Roy had but it looks like it might top off around 32k.

But…. On DNS-Operations

Peter van Dijk:

I now have ~130k (different!) keys, with 32201 unique key tags. This is almost twice as much as Roy had but it looks like it might top off around 32k.

Now what…

Three different tools

Two different libraries

Three issues:

1) Not enough keytags (expected 64K, got less)
2) Off by a few keytags (16387, 16385, 32769)
3) One library produces 50% of the other library

Is it the keytag function?

The keytag function is very similar to a radix minus one complement function. Very similar to the Internet Header Checksum.

So, generate 2048 random bits in pairs of 2 byte words and do an Internet Header Checksum over that.

```
while true
   do jot -r 128 0 65535 | awk \
   '{s+=$1} END {print (s + int(s/65536))%65535}’ \
   >>test
done

sort –u test | wc –l
65536
```

It is not the keytag function

The keytag function is very similar to a radix minus one complement function. Very similar to the Internet Header Checksum.

So, generate 2048 random bits in pairs of 2 byte words and do an Internet Header Checksum over that.

- It is not the keytag function

It is not the keytag function

The keytag function is very similar to a radix minus one complement function. Very similar to the Internet Header Checksum.

So, generate 2048 random bits in pairs of 2 byte words and do an Internet Header Checksum over that.

- It is not the keytag function
- It is not the library

It is not the keytag function

The keytag function is very similar to a radix minus one complement function. Very similar to the Internet Header Checksum.

So, generate 2048 random bits in pairs of 2 byte words and do an Internet Header Checksum over that.

- It is not the keytag function
- It is not the library
- It is not the tools

It is not the keytag function

The keytag function is very similar to a radix minus one complement function. Very similar to the Internet Header Checksum.

So, generate 2048 random bits in pairs of 2 byte words and do an Internet Header Checksum over that.

- It is not the keytag function
- It is not the library
- It is not the tools

(and hopefully not the user)

?

Florian Maury and Jérôme Plût

The Internet Header Checksum is equivalent to

addition modulo 65535

Florian Maury and Jérôme Plût

The Internet Header Checksum is equivalent to

addition modulo 65535

Assuming a 32 bit number `($num)` this means:

```
($num AND 65535) + ($num >> 16)
```

is equivalent to

```
$num % 65535
```

Florian Maury and Jérôme Plût

```
$num % 65535
```

In our case, $num contains the RDATA of the DNSKEY.

For all the keys generated, the RDATA part contains a constant:

(RDLENGTH,PROTOCOL,ALGORITHM, EXPONENT)

And a variable part:

The RSA modulus, which consist of two prime factors P and Q

Florian Maury and Jérôme Plût

Therefore, we have

```
$num % 65535
```

Is equivalent to:

```
(constant + P*Q) % 65535
```

Is equivalent to:

```
(constant % 65535) + ((P*Q) % 65535 )
```

Florian Maury and Jérôme Plût

Ignoring the constant part we have:

`(P*Q) % 65535`

We know that P and Q are very large primes.

65535 has factors: `3, 5, 17, 257`

Since `(P, Q, 3, 5, 17 and 257)` are co-prime,

`P, Q can’t be divided by 3, 5, 17 and 257`

and

`(P*Q) % 3, 5, 17 or 257 will never be 0`

Florian Maury and Jérôme Plût

```
(P*Q) % 3, 5, 17 or 257 will never be 0

(P*Q) % 3 has 2 solutions (not 3)
(P*Q) % 5 has 4 solutions (not 5)
(P*Q) % 17 has 16 solutions (not 17) and
(P*Q) % 257 has 256 solutions (not 257)

So, (P*Q) % 65535 has 2*4*16*256 solutions
```

Florian Maury and Jérôme Plût

```
(P*Q) % 3, 5, 17 or 257 will never be 0

(P*Q) % 3 has 2 solutions (not 3)
(P*Q) % 5 has 4 solutions (not 5)
(P*Q) % 17 has 16 solutions (not 17) and
(P*Q) % 257 has 256 solutions (not 257)

So, (P*Q) % 65535 has 2*4*16*256 solutions, or
```

32768 different keytags

Hoorah!

Three issues, one solved:

1) SOLVED: Not enough keytags (expected 64K, got less)
2) Off by a few keytags (16387, 16385, 32769)
3) One library produces 50% of the other library

Off by a few

Very similar is not exactly the same

The last part of the key-tag function in RFC4034 reads as follows:

```
ac += (ac >> 16) & 0xFFFF;
return ac & 0xFFFF;
```

If the previous line result in a carry (value > 65535), the latter line ignores it.

Hence, some off by a few keytags are a result of that.

Hoorah!

Three issues, two solved:

1) SOLVED: Not enough keytags (expected 64K, got less)
2) SOLVED: Off by a few keytags (16387, 16385, 32769)
3) One library produces 50% of the other library

Half the keyspace

Peter, using mbedTLS was able to produce twice as many keytags.

OpenSSL only generates safe primes:

P = 2 * P` + 1 where P` is also prime.

That implies that P mod 3 is never 1 (and thus always 2)

So: P*Q=M

(P mod 3) * (Q mod 3) = M mod 3
2 * 2 = 4 mod 3

M mod 3 is 1. Always

Half the keyspace

```
(P*Q) % 3, 5, 17 or 257 will never be 0

(P*Q) % 3 has 2 solutions (not 3)
(P*Q) % 5 has 4 solutions (not 5)
(P*Q) % 17 has 16 solutions (not 17) and
(P*Q) % 257 has 256 solutions (not 257)

So, (P*Q) % 65535 has 2*4*16*256 solutions, or
```

32768 different keytags

Half the keyspace

```
(P*Q) % 3, 5, 17 or 257 will never be 0
(P*Q) % 3 will always be 1
(P*Q) % 3 has 1 solution  (not 3)
(P*Q) % 5 has 4 solutions (not 5)
(P*Q) % 17 has 16 solutions (not 17) and
(P*Q) % 257 has 256 solutions (not 257)

So, (P*Q) % 65535 has 1*4*16*256 solutions, or
```

~~32768~~ different keytags

16384

Hoorah!

Three issues, two solved:

1) SOLVED: Not enough keytags (expected 64K, got less)
2) SOLVED: Off by a few keytags (16387, 16385, 32769)
3) SOLVED: One library produces 50% of the other library

Thanks to

Warren Kumari
Ben Laurie
Florian Maury
Jérôme Plût
Jean-René Reinhard
Peter van Dijk
Bert Hubert
David Conrad

And all who have participated in the discussions on dns-operations

Questions?