

DNSSEC in 6 minutes

- Update History

Un-numbered - Initial Release

1.1 - Grammar Corrections, added version number

1.2 - Split into 2 parts

1.3 - Correction in `dnssec-keygen`, added update history

1.4 - Correction of DLV

1.5 - Cleanup of split and updates to `udp53.org`
baseline version for Chinese translation

Thanks to...

- Francis DuPont
- Mark Andrews
- Tim Brown
- Håvard Eidnes
- Bruce Esquibel
- Carl Byington
- 孙国念 (SUN Guonian)

DNSSEC in 6 minutes

Alan Clegg
Support Engineer

Internet Systems Consortium

alan_clegg@isc.org

Version 1.5



Understanding DNSSEC

Understanding DNSSEC

- DNSSEC enabled authoritative servers provide digital signatures across RRsets in addition to “standard” DNS data
- DNSSEC validating resolvers provide authenticated responses with proven integrity

Understanding DNSSEC

- Clients using validating resolvers get guaranteed “good” data
 - for some value of “guaranteed”
- Data that does not validate provides a “SERVFAIL” response from the upstream resolver

Deploying DNSSEC

Deploying DNSSEC

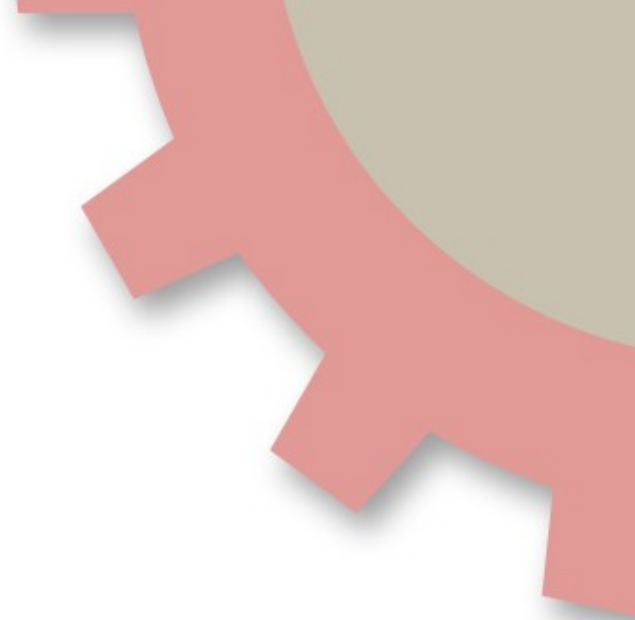
- One-time activities:
 - Clarify authoritative server directory structure and zone file naming
 - Enable DNSSEC on authoritative servers
 - Enable DNSSEC on recursive servers

Deploying DNSSEC

- DNSSEC enable each zone
 - Generate ZSK and KSK
 - Include keys into zonefile
 - Sign the zone
 - Point `named.conf` at the signed zonefile
 - Reload zone

Deploying DNSSEC

- Provide parent zone with DS records
- In the case of a DNSSEC unaware parent, provide DLV registry with DLV records



All of those steps...
in detail!

Prepare directory structure

- Tools are available that make zone maintenance easy but they work best with a standardized directory structure
- Put all files for a zone into a single directory

Enable authoritative servers

```
options {  
    dnssec-enable yes;  
};
```

- Requires BIND to have been built on a system with OpenSSL libraries available

Enable recursive servers

```
options {  
    dnssec-enable yes;  
    dnssec-validation yes;  
};
```

- Validation is done on the recursive, not authoritative servers.

Securing a Zone

- For each zone, two keys are created
 - 1) Zone Signing Key – used to sign the data within the zone
 - 2) Key Signing Key – used to sign the Zone signing key and to create the “Secure Entry Point” for the zone

Create the Keys

- Creating the ZSK

```
dnssec-keygen -a RSASHA1 -b 1024  
-n ZONE zonename
```

- Uses the RSASHA1 algorithm
- 1024 bits in length
- This is a DNSSEC ZONE key

Create the Keys

- Creating the ZSK

```
dnssec-keygen -a RSASHA1 -b 1024  
-n ZONE zonename
```

- Creates 2 files

```
Kzonename+<alg>+<fing>.key
```

```
Kzonename+<alg>+<fing>.private
```

- .key is public portion of the key
- .private is private portion of the key

Create the Keys

- Creating the KSK:

```
dnssec-keygen -a RSASHA1 -b 4096  
-n ZONE -f KSK zonename
```

- Uses the RSASHA1 algorithm
- 4096 bits in length
 - This large key will need lots of entropy!
- This is a DNSSEC ZONE key
- Has the Secure Entry Point (KSK) bit set

Prepare the Zone

- Add the public portions of both KSK and ZSK to the zone to be signed

```
$INCLUDE in zonefile or
```

```
cat Kzonename+*.key >> zonefile
```

- Beware of only using one > !

Sign the Zone

- Add the RRSIG, NSEC and associated records to the zone

```
dnssec-signzone [-o zonename]
                [-N INCREMENT] [-k KSKfile]
                zonefile [ZSKfile]
```

- zonename defaults to zonefile
 - Name the file after the zone!

Sign the Zone

```
dnssec-signzone [-o zonename]  
                [-N INCREMENT] [-k KSKfile]  
                zonefile [ZSKfile]
```

- -N INCREMENT automatically increments the serial number during signing
 - Removes “human error factor”

Sign the Zone

```
dnssec-signzone [-o zonename]
                [-N INCREMENT] [-k KSKfile]
                zonefile [ZSKfile]
```

- KSKfile defaults to Kzonefile*
 - with SEP bit set
- ZSKfile defaults to Kzonefile*
 - without SEP bit set

Sign the Zone

```
dnssec-signzone [-o zonename]
                [-N INCREMENT] [-k KSKfile]
                zonefile [ZSKfile]
```

- Output file is `zonefile.signed`
 - Sorted in alphabetical order
 - RRSIG, NSEC & DNSKEY RRs included
 - Much larger than before!

Update named.conf

Replace

```
zone "zonename" {  
    file "dir/zonefile";  
};
```

With

```
zone "zonename" {  
    file "dir/zonefile.signed";  
};
```


Start serving signed zone

- Tell named to re-read the configuration

```
rndc reconfig  
rndc flush
```

- You are now serving DNSSEC signed zones



Periodic Maintenance Issues

Periodic Zone Maintenance

- Signatures have lifespans
 - “Born-on” date – 1 hour prior to running `dnssec-signzone`
 - Expiration date – 30 days after running `dnssec-signzone`
- Expired signatures lead to zones that will not validate!

Periodic Zone Maintenance

- Any time you modify a zone – or at least every 30 days (minus TTL) you must re-run `dnssec-signzone`
- If you don't
 - 1) Zone data will be stale
 - 2) Zone data will be GONE

Periodic Key Maintenance

- Keys need to be rotated
 - No “expiration date”
- The longer a key is in public view, the more likely it is to be compromised
- Compromise (theft) of a key may lead to the need to “roll” a key over

Periodic Key Maintenance

- KSK should be rolled once a year
- ZSK should be rolled every 3 months
- Procedure is more complex than this presentation will get into
- Automation exists now!

Real-World Example

Sample with real names

- zonename to sign is `udp53.org`
- zonefile name is `udp53.org`
- Directory containing zonefile is `/zone/udp53.org`

Full path to zonefile is:

`/zone/udp53.org/udp53.org`

Sample with real names

```
<add dnssec-enable to named.conf>
```

```
cd /zone/udp53.org
```

```
dnssec-keygen -a rsasha1 -b 1024 -n ZONE  
udp53.org
```

```
dnssec-keygen -a rsasha1 -b 4096 -n ZONE  
-f KSK udp53.org
```

```
cat Kudp53*key >> udp53.org
```

```
dnssec-signzone -N INCREMENT udp53.org
```

```
<change zone file entries to use .signed>
```

Sample with real names

- Initially, /zone/udp53.org contained ONLY the zonefile "udp53.org"
- When finished:
 - 2 zsk files (.key and .private)
 - 2 ksk files (.key and .private)
 - 2 zonefiles (unsigned and .signed)
 - dsset-udp53.org file (DS RRs)
 - keyset-udp53.org file (DNSKEY RRs)

Sample with real names

- `zonefile` began with
 - 71 lines
 - 2,378 characters
- Ended with
 - 665 lines
 - 26,970 characters

Notify parent of DNSSEC

- Your parent zone must now insert a “DS” RR to create a chain-of-trust
- Procedures will differ between organizations, but this must be done securely
 - will require use of dsset- and/or keyset- files

DNSSEC unaware parent

- Not all TLDs support DNSSEC
 - Actually, **VERY FEW** TLDs currently support DNSSEC
- Provide your `DNSKEY` to those that you wish to have validate your zone
 - This must be done securely, not just with “`dig`”

Trust Anchors

Trust Anchors

- To validate other zones, you must insert “trust anchors” for each zone apex below which you wish to validate
- The ultimate trust anchor would be a signed DNS root (“.”) with fully populated TLDs

Trust Anchors

- When the DNS root (“.”) is signed, there will only be one required trust anchor
- Even after the DNS root is signed, it is still possible and probably necessary to have additional trust anchors

Trust Anchors

- At this time (Summer 2008), the DNS root (".") isn't signed
- Individual trust-anchors are required
- Trust anchors must be obtained by trusted means
- DNS is not one of those means, **HOWEVER...**

Trust Anchors

```
dig udp53.org DNSKEY
```

```
udp53.org. 14400 IN  DNSKEY 256 3 5 BE[...]/V1  
udp53.org. 14400 IN  DNSKEY 257 3 5 BE[...]|y|ot7
```

- Doing the “dig” provides something that can be verified via other means (web, phone, printed media, etc.)

Trust Anchors

- `named.conf` will need to contain:

```
trusted-keys {  
    "udp53.org." 257 3 5 "BE[...]1y1ot7";  
    "isc.org." 257 3 5 "BEAAAAO[...]ZCqoif";  
};
```

- An entry for EVERY zone apex below which you wish to validate

Trust Anchors

- Individual trust anchors do not scale well
- To help solve this problem, ISC created the DLV “Domain Lookaside Validation” RR and registry concept



Domain Lookaside Validation

DLV

- When validating, a resolver looks in the parent zone for a DS record for the zone being validated
- If it does not exist, a query for a DLV record in the DLV registry zone is made
- If successful, the DLV RR is used as the DS for the given zone

DLV Example

- `udp53.org` is signed
- The owner of `udp53.org` has registered with ISC's DLV Registry
- A DNSSEC query is made for the A RR for the label www.udp53.org
- No DS record is found in `.org` for the `udp53.org` zone

DLV Example

- A non-DLV enabled recursor will not be able to do validation at this point
- A DLV enabled recursor will look for `udp53.org.dlv.isc.org`. DLV RR
- That DLV RR will then be used as the DS for the `udp53.org`. zone

Enabling DLV

- Use of DLV to validate is done on the recursive server
 - A trust anchor must be created for the DLV registry
 - `dnssec-lookaside` must be linked to the DLV trust anchor

Enabling DLV

- `named.conf`:

```
trusted-keys {
    dlv.isc.org. 257 3 5 "BEA[...]uDB";
};
options {
    dnssec-lookaside "."
    trust-anchor dlv.isc.org.;
};
```

Generating DLV RRS

- When signing a zone for a DLV registrar, add the “-1” (ell) switch to `dnssec-signzone`:

```
dnssec-signzone [-o zonename]
[-N INCREMENT] -l dlvzone
[-k KSKfile] zonefile [ZSKfile]
```

- `dlvzone` will be registrar dependent

Generating DLV RRS

- Based on the previous example:

```
dnssec-signzone -N INCREMENT  
-1 dlv.isc.org. udp53.org
```

- At this point, the file `dlvkey-udp53.org` will be created and ready to send to the ISC DLV administrator

Registering with DLV

- Contact the DLV registrar for instructions on how to prove ownership of zone and validity of DLV RR
- Insertion of your DLV RR into the DLV registry must be done in a trusted manner

ISC's DLV registry

<http://www.isc.org/ops/dlv/>



Questions?

Comments?



No! No, no, not 6!

I said 7

Nobody's comin'
up with 6

Who deploys
DNSSEC in 6
minutes?

Testing and Debugging DNSSEC

Testing DNSSEC

- Now that you are distributing DNSSEC signed RRsets, is it working?
- Mark Andrews stated that DNSSEC can be debugged using only “dig” and “date”
- Here's how!

digging DNSSEC

- A query asked for valid data from any recursor will provide the RRset in response
- A query asked for non-signed data from any recursor will provide the RRset in response

digging DNSSEC

- A query asked of a validating recursor for modified or invalid data will return `SERVFAIL`
- Applications (and users) will see this as domains that “vanish”
- A header bit (CD) will allow invalid data to be passed anyway

dig output – no DNSSEC

```
dig www.udp53.org a
```

```
;; [..] status: NOERROR
```

```
;; flags: qr rd ra;
```

- Good answer; Response, Recursion Desired, Recursion Available

dig output – no DNSSEC

```
dig www.udp53.org a
```

```
;; [...] status: NOERROR
```

```
;; flags: qr rd ra;
```

- From a validating recursive, this is guaranteed good data

dig output – no DNSSEC

```
dig www.udp53.org a
```

```
;; [..] status: NOERROR
```

```
;; flags: qr rd ra;
```

- But how do you know that your recursor is doing validation?

dig output – DNSSEC

```
dig +dnssec www.udp53.org a
```

```
;; [..] status: NOERROR
```

```
;; flags: qr rd ra ad
```

- As before, but this time, Authenticated!

digging DNSSEC

- To return AD set, the validating recursor must have a trust anchor that can be tracked back to (via DS RRs)
- If the chain of trust does not lead to a trust anchor, AD will not be set but RRSIG RRs will still be returned

dig output – DNSSEC

```
dig +dnssec www.udp53.org a
```

```
www.udp53.org. 3600 IN A      192.168.154.2
www.udp53.org. 3600 IN RRSIG  A 5 3 3600 20080627122225
20080617122225 46704 udp53.org.
XEKXkv9MCRiGbxO9T0dkNY+3y5EZRB6s6YOk0pFAVUL/y8VDeJphc8yb
K6E/YLvraItGvIvpy4P1OuIY09BGQ==
```

- If AD is set, recursor tracked back to a trust anchor, if not, we still have data that we can validate ourselves

digging DNSSEC

- If we know that we are talking to a validating recursor, and we get `SERVFAIL`, it may be non-validating signed data
- If so, setting the “CD” bit in the query will cause the recursor to send the “bad” data anyway

dig output – DNSSEC

```
dig +dnssec +cd www.udp53.org a
```

```
www.udp53.org. 3600 IN A      192.168.154.2
```

```
www.udp53.org. 3600 IN RRSIG A 5 3 3600 20080627122225  
20080617122225 46704 udp53.org. xxxxxxxxxxx
```

- Invalid RRSIG (**xxxxxxxxxxx**), but with +cd, we get a response anyway

dig output – DNSSEC

```
dig +dnssec +cd www.udp53.org a
```

```
www.udp53.org. 3600 IN A      192.168.154.2
www.udp53.org. 3600 IN RRSIG  A 5 3 3600 20030627122225
20030617122225 46704 udp53.org.
XEkXkv9MCRiGbxO9T0dkNY+3y5EZRB6s6YOk0pFAVUL/y8VDeJphc8yb
K6E/YLvraIt dGvIvpy4P1OuIY09BGQ==
```

- Dates in signature show that it has expired
- Compare with “date”

digging DNSSEC

- Note that it is easy to check the date on the signatures
- It's much harder (humanly impossible?) to find an error in the key itself
- The previous example is extremely contrived (xxx?)

digging DNSSEC

- Another problem that can occur is a missing hash or key
 - DS in parent
 - DNSKEY in current zone
- Not hard to determine this fault either!

dig output – DNSSEC

```
dig +dnssec +cd www.udp53.org
```

```
www.udp53.org. 3600 IN A      192.168.154.2
www.udp53.org. 3600 IN RRSIG  A 5 3 3600 20080627122225
20080617122225 46704 udp53.org.
XEkXkv9MCRiGbxO9T0dkNY+3y5EZRB6s6YOk0pFAVUL/y8VDeJphc8yb
K6E/YLvraItGvIvpy4P1OuIY09BGQ==
```

- This signature was created with key 46704

dig output – DNSSEC

```
dig +cd +multi udp53.org dnskey
```

```
udp53.org. 14400 IN DNSKEY 256 3 5 (  
  BEAAAAO2oQi7U9m9i495S/XoAk+j8QxxnBHon6fa7nlN  
  7xoqrSr/xzy3+IerFS1KgJz1gJGbTsGV0WI1/bvAzIEK  
  Uh+p ) ; key id = 46704
```

- DNSKEY in zone exists
- If not, it won't validate!

dig output – DNSSEC

```
dig +cd +multi udp53.org dnskey
```

```
udp53.org. 14400 IN DNSKEY 256 3 5 (  
  B[...]p ) ; key id = 46704
```

```
udp53.org. 14400 IN DNSKEY 257 3 5 (  
  B[...]J ) ; key id = 64249
```

- ZSK DNSKEY in zone exists
- Associated KSK is 64249

dig output – DNSSEC

```
dig +norec @gTLD udp53.org ds
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29385  
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 2
```

- DS does not exist in parent
- If we don't do DLV, this is why it won't authenticate

dig output – DNSSEC

```
dig +norec @gTLD udp53.org ds
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29385  
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 2
```

- Server will still return “non-AD” DNSSEC data
- Because there is no chain-of-trust to a trust-anchor

dig output – DNSSEC

```
dig udp53.org.dlv.isc.org dlv
```

```
udp53.org.dlv.isc.org. 3257 IN DLV 64249 5 2 (  
 59C58FD329F1C33628C92FC4B763EF9ADB833804D60D  
 18D439AB04F6302C20FD )
```

```
udp53.org.dlv.isc.org. 3257 IN DLV 64249 5 1 (  
 D5D722703D848E85D85E8A8442AF47512B385418 )
```

- **KSK 64249 DLV** does exist in ISC's registry, providing DS for zone

digging DNSSEC

- Trust anchor for `dlv.isc.org`
- DLV entry for `udp53.org.dlv.isc.org`
- KSK for `udp53.org`
- ZSK for `udp53.org`
- Signature for www.udp53.org
- AD bit set!



Questions?

Comments?