DNSSEC Tutorial

Shumon Huque
bdNOG 11, Bangladesh
January 11th 2020
Recursive Resolver is prepopulated with root DNS server addresses.

The resolver (recursive server) handles the query for `www.amazon.com` by recursively querying the root DNS server.

2. The root DNS server refers to the .com domain.
3. The resolver queries the .com domain.
4. The .com domain refers to the amazon.com domain.
5. The resolver queries the amazon.com domain.
6. The amazon.com domain answers with the IPv4 address 1.2.3.4.
7. The resolver returns the answer to the endstation.
8. The endstation then uses the IP address to communicate with the Amazon website.

This process demonstrates how recursive resolvers work to resolve domain names to IP addresses.
Why DNSSEC?
Plain DNS isn’t secure

- DNS responses are easily spoofed
- On-path attacks are trivial
- Even off-path (blind) attacks are often easy
- One UDP packet for query, one for response
- Must rely on source address and query IDs to match response to query (often not hard to guess)
- We need real security built on cryptography --&gt; DNSSEC
DNSSEC at a glance

• “DNS Security Extensions”
• A system to verify the authenticity of DNS “data” using public key signatures
• Helps detect DNS spoofing, cache poisoning ..
• Additional benefits:
  • Ability to store and use security keys in the DNS
(Some) Crypto 101
Whose heard of any of the following?

RSA
ECDSA
SHA1
SHA256
RSASHA1
RSASHA256
ECDSAP256SHA256
Public Key Encryption/Signature Algorithms:

RSA
ECDSA

Cryptographic hash functions:

SHA1
SHA256

Combined modes of using them to make signatures:

RSASHA1
RSASHA256
ECDSAP256SHA256
Traditional Cryptography

• “Symmetric Key Cryptography”

• The same key is used to both encrypt and decrypt data

Encryption

Decryption
Public Key Cryptography

• Aka “Asymmetric Key Cryptography”

• Based on some mathematical problems that are difficult to solve efficiently:
  • factoring large numbers, computing discrete logarithms, etc.

• A pair of mathematically related keys

• Public and Private (or Public and Secret)

• Used in Encryption and Signing

• In DNSSEC, we only care about signing
Public Key Cryptography

• A pair of mathematically related keys: public and private
• Public key: used to Encrypt data or Verify signatures
• Private key: used to Decrypt data or Sign data

Encryption

Decryption
Public Key Cryptography

• A pair of mathematically related keys: public and private

• Public key: used to Encrypt data or Verify signatures

• Private key: used to Decrypt data or Sign data

Signing

Verifying signatures
RSA Cryptosystem

- Rivest, Shamir, Adleman, 1977 - first practical public key cryptosystem for encrypting/decrypting data
- Based on practical difficulty of factoring the product of two large prime numbers
- Derives encryption & decryption exponent based on these primes
- Encryption/decryption operations are exponentiation modulo the product of the primes
- Breaking the system requires computing e’th roots, which cannot be done efficiently without knowing the two primes
RSA Cryptosystem

Pick large primes, $p$ and $q$, (~ 1024 bits)
The RSA modulus, $N = p \times q$ (~ 2048 bits)
Choose integers $e$, $d$ such that,
\[ e \times d \equiv 1 \pmod{(p-1)(q-1)} \]

- $e$ is the public exponent
- $d$ is the private (secret) exponent

Public key = $(N, e)$  -> published to world
Private key = $(p, q, d)$  -> kept secret by owner

Given input data $x$,
Sign function: $y = x^d \mod N$
Verify function: $y^e \mod N$; does it equal $x$?

(Text book RSA; actual RSA is a bit more complicated)
RSA Cryptosystem

Pick large primes, $p$ and $q$, (~ 1024 bits)

The **RSA modulus**, $N = p \times q$ (~ 2048 bits)

Choose integers $e$, $d$ such that,

$$e \times d = 1 \mod (p-1)(q-1)$$

- $e$ is the public exponent
- $d$ is the private (secret) exponent

Public key = $(N, e)$  

Private key = $(p, q, d)$

-> published to world  

-> kept secret by owner

Given input data $x$,

Signing function: $y = x^d \mod N$

Verify function: $y^e \mod N$; does it equal $x$?

*(Text book RSA; actual RSA is a bit more complicated)*
Cryptographic Hash Functions

• Compression function
  • Generates fixed size output from arbitrary sized input data

• Not reversible (One-way)

• Collision resistance
  • Computationally difficult to find another input that generates the same hash output (or any 2 inputs that hash to the same value)
  • Obviously collisions exist, but are impractical to find

• Examples: SHA1, SHA256, SHA512 etc
Cryptographic Hash Functions

IP address of verisign.com is 69.58.181.89

Hash Function

41af286dc0
Cryptographic Hash Functions

IP address of verisign.com is 69.58.181.89

IP address of verisign.com is 66.66.66.66
Non-reversible ("pre-image resistance"): given a hash output, it's difficult (i.e. impractical) to figure out what the input was. "One way function."

However, if someone tells you the input, you can easily verify that the hash output is correct. (by running the input through the hash function)
Cryptographic Hash Functions

IP address of verisign.com is 69.58.181.89

???

Weak Collision Resistance ("2nd pre-image resistance"): given an input and its hash, difficult to find a second input that produces the same hash.
Cryptographic Hash Functions

Strong Collision Resistance: difficult to find any 2 inputs (A and B) that produce the same hash output (h)
We sign hashes of data

• Generally, we don't sign the data directly

• We sign hashes of the data

• Hashes are non-invertible and collision resistant, hence we can treat the hashes as unforgeable representations of the data

• Hashes can be computed very fast

• Computationally more efficient to sign small sized hashes than data which could be arbitrarily large

• The hashes are fixed size, so they produce fixed size signatures
Signing and Verifying

data → H → hash

Signing data

Verifying signatures
Signing and Verifying

Signing data

Verifying signatures

data → H → hash

hash → E → signature

private key
Signing and Verifying

1. Hash the data:
   \[ H \text{ (data)} \rightarrow \text{hash} \]

2. Encrypt the hash with the private key:
   \[ \text{hash} \rightarrow E \rightarrow \text{signature} \]

   - private key

Send: data + signature

Signing data

Verifying signatures
Signing and Verifying

Send:

- **data** + **signature**

**Signing data**

Receive:

- **data** + **signature**

**Verifying signatures**

- **data** $\rightarrow$ **H** $\rightarrow$ **hash**
- **hash** $\rightarrow$ **E** $\rightarrow$ **signature**
- **private key** $\uparrow$

**Sign**:

- **hash** $\rightarrow$ **signature**

**Verify**:

- **data** $\rightarrow$ **hash**
- **signature** $\rightarrow$ **signature**
Signing and Verifying

Send:
- Data + Signature

Signing Data

Receive:
- Data + Signature

Verifying Signatures

1. Hash Data
2. Encrypt Hash with Private Key
3. Send Data + Signature

1. Hash Data
2. Compare Original Hash with Hash of Received Signature
Signing and Verifying

Send:

Data + signature

Signing data

Receive:

Data + signature

Verifying signatures
Signing and Verifying

Send:

Data + Signature

Signing data

Receive:

Data + Signature

Verifying signatures

Are these hashes the same?
DNSSEC at a glance

• “DNS Security Extensions”

• A system to verify the authenticity of DNS “data” using public key signatures
  • Specs: RFC 4033, 4034, 4035, 5155, 6840

• Helps detect DNS spoofing, misdirection, cache poisoning ..

• Additional benefits:
  • Ability to store and use cryptographic keying material in the DNS, eg. SSHFP, IPSECKEY, CERT, DKIM, TLSA, etc ..
DNSSEC at a glance

• Each zone has a public and private key pair

• The zone owner uses the private key to sign the zone data, producing digital signatures for each resource record set

• Public key is used by others (DNS resolvers) to validate the signatures (proof of authenticity)

• Public key is published in the zone itself so that resolvers can find it

• Zone public keys are organized in a chain of trust following the normal DNS delegation path

• DNS resolvers authenticate DNS signatures from root to leaf zone containing name.
What DNSSEC doesn't do

• Does not provide confidentiality
  • Data is not encrypted, only signed (integrity protection)
  • Most data in the DNS are public anyway
• Does not provide transport security
• Does not address other classes of attacks such as denial of service, software bugs, etc
## DNSSEC Records

<table>
<thead>
<tr>
<th>DNSKEY</th>
<th>Contains zone public key</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRSIG</td>
<td>Contains DNSSEC signature</td>
</tr>
<tr>
<td>NSEC</td>
<td>Points to next name in zone (used for authenticated denial of existence)</td>
</tr>
<tr>
<td>DS</td>
<td>Delegation Signer (certifies public key for a child zone)</td>
</tr>
<tr>
<td>NSEC3</td>
<td>Enhanced version of NSEC (provides zone enumeration protection and opt-out)</td>
</tr>
<tr>
<td>NSEC3PARAM</td>
<td>NSEC3 parameters</td>
</tr>
</tbody>
</table>
Signed zone additions

• One or more DNSKEY at the zone apex
• One or more NSEC for every DNS name
• One or more RRSIG for every RR set
• One or more DS records for every secure delegation

• Exceptions: non-authoritative data like delegation NS records and glue have no signatures (RRSIG)
Multiple DNSKEYs

• Typically, a 2-level hierarchy of DNSKEYs is employed

• KSK: Key Signing Key
  • Signs other keys (can be larger, ie. stronger, and kept offline; used as the trust anchor and certified by the parent zone in the DS)

• ZSK: Zone Signing Key
  • Signs all other data in the zone (can be lower strength and impose less computational overhead; can be changed without co-ordination with parent zone)
Protection of signing keys

• Keep offline? But poses problems for dynamic signing

• Keep only KSK offline? But need to bring them online for key rollovers (even only ZSK rollovers)

• If keeping online, lock down housing server rigorously, as you might do a critical authentication server, like a KDC

• Physically secured machine room & racks

• Tamper resistant HSM (Hardware Security Module)
Signing and Verifying

Send:

- data + signature
  
  Signing data

Receive:

- data + signature
  
  Verifying signatures

- data
  
  hash
  
  signature

- signature
  
  hash

- public key

Are these hashes the same?
Signing and Verifying

**Send:**
- **DNS RRset** data
  - SHA256 hash
  - RSA hash
  - **private key**
    - +PKCS1 encoding

**Receive:**
- **data** + **signature**
  - DNS RRset
  - SHA256 hash

**Signing data**
- **data** + **signature**
  - DNS RRset
  - DNS RRSIG

**Verifying signatures**
- **signature**
  - DNS RRSIG
  - RSA hash

Are these hashes the same?
What does a DNS response with DNSSEC signatures look like?
$ dig jabber.upenn.edu A

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 337

;; QUESTION SECTION:
;jabber.upenn.edu. IN A

;; ANSWER SECTION:
jabber.upenn.edu. 86400 IN A 128.91.2.172

;; AUTHORITY SECTION:
upenn.edu. 86400 IN NS sns-pb.isc.org.
upenn.edu. 86400 IN NS adns3.upenn.edu.
upenn.edu. 86400 IN NS adns2.upenn.edu.
upenn.edu. 86400 IN NS adns1.upenn.edu.
upenn.edu. 86400 IN NS dns2.udel.edu.
upenn.edu. 86400 IN NS dns1.udel.edu.

;; ADDITIONAL SECTION:
adns1.upenn.edu. 81904 IN A 128.91.3.128
adns1.upenn.edu. 81904 IN AAAA 2607:f470:1001::1:a
adns2.upenn.edu. 81904 IN A 128.91.254.22
adns2.upenn.edu. 81904 IN AAAA 2607:f470:1002::2:3
adns3.upenn.edu. 81904 IN A 128.91.251.33
adns3.upenn.edu. 81904 IN AAAA 2607:f470:1003::3:c

RRset without signature
$ dig jabber.upenn.edu A +dnssec

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 690
;; flags: qr aa rd ad
QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 7

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: udp: 4096
;; QUESTION SECTION:
jabber.upenn.edu. IN A

;; ANSWER SECTION:
jabber.upenn.edu. 86400 IN A 128.91.2.172
jabber.upenn.edu. 86400 IN RRSIG A 5 3 86400 
  (20140909070510 20140810061231 50475 upenn.edu. jFElfhdkNeWSIwEZCv/Vn2nt9T2KXXkogYYtyemVEf3X4I4nbhyXGvFdGEmDS9cVl3RgiwwUVaY78jaz7cQPX21c3raYDrLV3irRh1NSbt9v/esF4SI06KwrhTv3Z2GBVP+CrjFkMLJnN1dFBEa2UHzFzIkk7cKoQcdmEdbiDJ3Ag= )

;; AUTHORITY SECTION:
upenn.edu. 86400 IN NS dns1.udel.edu.
upenn.edu. 86400 IN NS noc3.dccs.upenn.edu.
upenn.edu. 86400 IN NS dns2.udel.edu.
upenn.edu. 86400 IN NS noc2.dccs.upenn.edu.
upenn.edu. 86400 IN RRSIG NS 5 2 86400 20140919232217 
  (20140819223616 50475 upenn.edu. WWpT4uD9p5zORM+207pRZ46+Qo3cHj9tnjxH62Xt9QBRyu9V7+3ihlIM1HCd9kjsdltkT8GJ+5hEzyk8fPJjsli
bqG6hCncCccGdTsGzmPoaGdlz95H7Nf2yfr1GLAcSCix6I
  EJb8A4j4+0W9Zq1dmeZrnJDxSm8joQg5+IlkzR4= )

(Algorithm 5: RSA-SHA1)
The image contains a diagram explaining the structure of a DNS RRSIG record. The key components are labeled as follows:

- **RRSIG**
- **Record Type Covered**
- **Algorithm**
- **#Labels (minus root)**
- **Original TTL**
- **Signature Inception**
- **Key Tag**
- **Zone Name**
- **Signature Expiration**
- **Cryptographic Signature Base64 Encoded**
- **signature of ( RRset + RRSIG_RDATA minus signature field)**

The diagram illustrates the process of creating a cryptographic signature for a DNS RRSIG record, which is used for verifying the authenticity of DNS data. The signature is derived from the RRset and RRSIG_RDATA, excluding the signature field.
Data that is signed

signature of ( RRset + RRSIG_RDATA minus signature field)
DNSKEY record

• Contains zone’s DNSSEC public key and associated flags

example.com. 7200 IN DNSKEY 256 3 5 (AwEAAfdhCwSImS/vqlPV7nQMPCCjd7P8za38PFH606zhz7zc7lirVewUphYijpDTeex2nEfq3leFKmHqQw4Oc7Jx
p7/Bdfr2uZfeCws0zwal2kZDJX/0+wBNqtIltc6tdwzX
tGU21VEvDsd13xiQpRHkxt9PclVlqnGv39hK9mimx7b)
; ZSK; alg = RSASHA1; key id = 50475
## Common DNSSEC algs

<table>
<thead>
<tr>
<th>Algorithm#</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>RSA-SHA1</td>
</tr>
<tr>
<td>6</td>
<td>DSA-NSEC3-SHA1</td>
</tr>
<tr>
<td>7</td>
<td>RSA-NSEC3-SHA1</td>
</tr>
<tr>
<td>8</td>
<td>RSA-SHA256</td>
</tr>
<tr>
<td>10</td>
<td>RSA-SHA512</td>
</tr>
<tr>
<td>12</td>
<td>ECC-GOST</td>
</tr>
<tr>
<td>13</td>
<td>ECDSA Curve P-256 SHA256</td>
</tr>
<tr>
<td>14</td>
<td>ECDSA Curve P-384 SHA384</td>
</tr>
</tbody>
</table>

$ dig upenn.edu DNSKEY

;; ANSWER SECTION:
upenn.edu. 7200 IN DNSKEY 256 3 5 (AwEAAcDtI07stSjvoBA/YVPr+2gvB3v33tXr7ROZ/JqmWtNLraxQPzgXM1AhwjtdeEqwCAnk01V7+Fw7K94sh6jpI 5bFofS7MGtd0VvNyq52bgRnusgbmlME2Lx9+o3fy9ppv 7C6bahGrV3aiq9wNVPj/ccJn5AnZCOsi3grVsj6izCYH ) ; key id = 46752
upenn.edu. 7200 IN DNSKEY 256 3 5 (AwEAAfAHsS33kJJEImVk09yFJY5hXumAo+JVVJMjpJUajl/rh0fFkdikS2oatVvxHHHqKN9Kg3DoKQss/CzCZa4znKlqYGvS17RefKR3QLyPBGN2aOUWxshDgOWLmOtqNpmpP+6Drfn8LJVT0juwmU80laQcdA/AoOGVPE3zP16G/F+qp ) ; key id = 43248
upenn.edu. 7200 IN DNSKEY 257 3 5 (AwEAAek95gyBF2nurd1E2Q63VvCMlaZOLQEnz0N4Ce89SB4JuW2eEBerpLmEauuGJbrsoOGx3SKCMyhOYL9q1ZrmCN6f6nACwv88NtrY0jHAOmlLVaKAFqv8MTBbEwTWBBw5K8jUwzcaGyDjo3U+Hai+ow8TievoBy+hrcT4DegsEBB8MEQIGeOo/Kw9wblJLEdpvVXtuv2178G75FUwMrA8jzEkaM7bKg/HSTIMupbwfs4IHYqBG/PkgOZYL3uxm9nCeVbjb4YYd4G6koVoWteWTS8JdYq4gr99AEjhwAzbe7bd7pX+qD70CCbh0jSOVhPvhRpfCHIYZAJIwEAWs711HHM= ) ; key id = 29242
Secure Delegations

• Indicated by DS (Delegation Signer) record
• Appears in the delegating (ie. parent) zone
• Contains a hash of the public key of the child zone’s
• Validating resolvers use the presence of the DS record and its corresponding signature (RRSIG) to securely authenticate the delegation
DS record

• “Delegation Signer” record
• Refers to & vouches for a DNSKEY in delegated child zone
• DNSKEY is usually the KSK or secure entry point key
• Contains a cryptographic digest (hash) of the DNSKEY
• 3 algorithms defined: 1=SHA1, 2=SHA256, 4=SHA384
A DS record in a parent zone refers to a specific DNSKEY record in a child zone.

**edu zone**

```plaintext
upenn.edu. 86247 IN DS 18463 5 2 ( 6003992326DA06785C9E30B259750FAB0960BF57054B DDFFDEEE1188977DABB8 )
86247 IN DS 18463 5 1 ( 0C45B3D090B221E0E33BBEB5A619D89416BAF197 )
86247 IN RRSIG DS 8 2 86400 ( 20131004044110 20130927033110 21638 edu. piBkiV626itFwzUdoKk8117ljbN2+EUcNYESuKShN2kw7GapeS0gTW5kafTtFlB7AjNN8AR85YrfH56XKwEvRool yN2Tz1EEeLtiQnMEwAi0FZ7NKz7eb2IImezGtFvUUqHmx0gf2B5r4JHFhyB74M10+BY16LWyedoujUsezD0w= )
```

**upenn.edu zone**

```plaintext
upenn.edu. 7200 IN DNSKEY 257 3 5 ( AwEAAF02wZZMtX2ofTKfJ/xoQffn17NFJV0Y5s4F3tMd kktC/abDax+SBOMJWRAcznigKoirdVL7ZbVS2DYRoUvJV v44VLRtLlbgxDnhpP4fCJ88Yu33/5GFZwmgxco4OA6y xhwEniIveQ5B7LJ0Vh8KyfqU6obu7wFR7pSV1UVybLZf F3n1Kb+6KRWtave5JLbhfYfXYxhUpWlbeKYmoro09SH sQaoR3vr7L168MEe4VRE+SKcuNRkzLBG5XQDnImYanv6 Pf4tYaNTGYPgkXSVJKeGUpncJ0xZ8NraQpGdQML3x ALGWHW1rpmene6EQdf7+qv1vm7uHxN1L0MhS7B4U= ) ; KSK; alg = RSASHA1; key id = 18463
```
Referral from .COM servers to verisigninc.com:

$ dig @a.gtld-servers.net +dnssec www.verisigninc.com.

;; AUTHORITY SECTION:


verisigninc.com. 86400 IN DS 64326 8 2 (02E7FEF4C3BBB0A0FA52F0F8E5774C44B243739D1AB7B3B426A417C388F45ACF)

verisigninc.com. 86400 IN RRSIG DS 8 2 86400 (20140907041542 20140831030542 6122 com. sgFME6uw21TtroVxPToD/do5u/Q5TBRzGFj8YcxE1h27vAHuvgLhSBf/QQVyRxIGHxBB0420jpuJzj2TgiHEgWa5CLxWS6izfhBT3+vZ5jzIn7ryNhV2rQD10iMk1tV4ncr/5PiHnHUFfGfUpprUgh/fT3C9nhb4v56xVxNHceI0= )}

NS Record Set
DS Record Set
Signature of DS records
Recursive Resolver is prepopulated with root DNS server addresses.

2. The resolver refers to .com.
3. .com
4. referral to amazon.com
5. amazon.com
6. answer 1.2.3.4
7. referral to .com
8. www.amazon.com

endstation (has stub resolver)
Recursive Resolver is prepopulated with root DNS server addresses and the root’s public key.

Validating Resolver
(has root’s pubkey)

www.amazon.com
set DO bit
referral to .com
+ DS, RRSIG

endstation
(has stub resolver)

www.amazon.com
answer + AD bit

. (root) root’s pubkey
Signed Root Zone

 Signed COM Zone
Signed Amazon Zone

. com
com pubkey

referral to amazon.com
+ DS, RRSIG

amazon.com
amazon pubkey
(Signed Amazon Zone
(Also queries for DNSKEY and DS records are performed as needed)

answer 1.2.3.4 + RRSIG

referral to .com
+ DS, RRSIG

1
8
answer + AD bit

2
3
4
5
6
7

53
In reality, more complex

• Most zones have the KSK, ZSK split/hierarchy
• So there is an additional level of signing and verifying going on at each delegation
• Zone operator uses KSK private-key to sign ZSK public key
• Resolvers have to
  • compare the DS to the child zone's corresponding KSK public key
  • then use the KSK pubkey to verify the signature on the ZSK pubkey
  • then use the ZSK pubkey to verify signatures on other data
Negative answers

• “Authenticated Denial of Existence”

• NSEC or NSEC3 records (and their signatures)

• Chain together DNS records in a zone; can think of them and their signatures as spanning the gaps between names in the zone

• Canonical ordering of names in signed zones needed (RFC 4034, Section 6.1)

• Needed because of the pre-computed signature model of DNSSEC (computational concerns & signing key security)
Canonical Order

Sort DNS names in order of most significant (rightmost) labels first. Then within each label, sort them as octet strings, case-folding ASCII letters to lowercase.

example.com
a.example.com
blah.a.example.com
Z.a.example.com
zABC.a.EXAMPLE.com
z.example.com\001.z.example.com
*.z.example.com\200.z.example.com

(See RFC 4034, Section 6.1)
NSEC record

• “Next Secure” record
• Describes interval between consecutive names in a zone
• Type-bitmap defines RRtypes available at owner
• Side Effect: allows enumeration of zone contents
An authenticated negative answer (NXDOMAIN)

$ dig +dnssec +multi bozo.upenn.edu A

;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 7708

;; ;; AUTHORITY SECTION:

[SOA and RRSIG(SOA) records omitted for brevity]

<table>
<thead>
<tr>
<th>Domain</th>
<th>Class</th>
<th>Type</th>
<th>Time to Live</th>
<th>Other Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>upenn.edu</td>
<td>3600</td>
<td>IN</td>
<td>NSEC</td>
<td>_kerberos.upenn.edu. NS SOA MX RRSIG NSEC DNSKEY TYPE65534</td>
</tr>
<tr>
<td>bozo.upenn.edu</td>
<td>3600</td>
<td>IN</td>
<td>RRSIG</td>
<td>NSEC</td>
</tr>
<tr>
<td>caggrid.bmif.upenn.edu</td>
<td>3600</td>
<td>IN</td>
<td>RRSIG</td>
<td>NSEC</td>
</tr>
</tbody>
</table>

bozo.upenn.edu would have been between caggrid.bmif.upenn.edu & brynmawr-gw.upenn.edu

*.upenn.edu would have been between upenn.edu and _kerberos.upenn.edu
An authenticated negative answer (nodata)

NOERROR (nodata) responses can be authenticated with one signed NSEC record, which just reports all available RRTYPEs at that name

$ dig +dnssec upenn.edu A

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44529

;; AUTHORITY SECTION:

[SOA and RRSIG(SOA) records omitted for brevity]

<table>
<thead>
<tr>
<th>Domain</th>
<th>Class</th>
<th>TTL</th>
<th>Type</th>
<th>Name</th>
<th>TimeStamp</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>upenn.edu</td>
<td></td>
<td>3600</td>
<td>NSEC</td>
<td>_kerberos.upenn.edu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>upenn.edu</td>
<td></td>
<td>3600</td>
<td>RRSIG</td>
<td>NSEC 5 2 3600 (</td>
<td>20120508051318 20120408042226 50475 upenn.edu. ZzTYjeHECy5xLo+wrXq1VwmtNI3Wz7cpNLBdg+3xM9ph H9jOndAViCKwsDa4uLBYBcKss9qbbYjVtMp5w0lmVpwm cwxYheAyEN+w2VPBhLZ9qjfib8Q6LfI3r31C8EDJciLi0 1LSQwP2gyFx7V6VG08z11W6fuB6A/6/3/55xwW0= )</td>
<td></td>
</tr>
</tbody>
</table>
**Wildcard proof?**

Positive wildcard responses includes NSEC RRs in AUTHORITY section to prove that the queried name did not explicitly exist in the zone.

Zone:
- ns2.example. 3600 IN A 192.0.2.2
- *.w.example. 3600 IN MX 1 ai.example.
- x.w.example. 3600 IN MX 1 xx.example.
  [etc]

;; Answer
- a.z.w.example. 3600 IN MX 1 ai.example.
- a.z.w.example. 3600 RRSIG MX 5 2 3600 20040509183619 (20040409183619 38519 example.
  OMK8rAZlepflzLWW75Dxd63jy2wsweSzxdKG2
  f9AMN1CytCdi0cYISAfAdvXZ7xujKAtPbc
tvQO2ofO7AZJ+d01EeeQTVBPq4/6KCWhqe2X
  TjnVtLVynvghnc0u28aoSsG0+4InvkkOHknKxxw
  4kX18MMR34i81C36SR5xBni8vHI= )

;; Authority
- x.y.w.example. 3600 NSEC xx.example. MX RRSIG NSEC
- x.y.w.example. 3600 RRSIG NSEC 5 4 3600 20040509183619 (20040409183619 38519 example.
  OvE6WUzN2zilieJcvKPwbCAyXyP6ef8cr6Csp
  ArVSTzKSuIvbezZmku7E34o5Imb6CWSSSpgr
  xv098kNUFnHcQf/LzY2zqRomubrNQhJTdTXX
  a0ArunJQCy2pOYq5t0SLjm6qp6McJ1AP5Vr
  QoKqJDCLnoAlcP0PKAm/jJkn3jk= )
NSEC3 differences

• NSEC3 instead of NSEC records

• Owner name is a cryptographic hash of the name (flattened) rather than the actual name - provides zone enumeration defense

• Some names may not have an NSEC3 (the “opt-out” feature)

• Additional apex record: NSEC3PARAM

• Increased CPU usage implications

• See RFC 5155 (Hashed Authenticated Denial of Existence) for details
NSEC3 record

- New version of NSEC that provides defense against zone enumeration (see RFC 5155 for details)
- Hashed owner names (base 32 with extended hex alphabet)
- Optional “opt-out” feature
- rdata: nsec3 parameters (hash alg, flags, iterations), hashed next owner name, type bitmap
NSEC3 hash chain
NSEC3PARAM record

- NSEC3PARAM record at zone apex also holds the parameters
- Hash algorithm, Flags, #Iterations, Salt
- This is used by secondary nameservers for the zone, to choose an appropriate set of NSEC3 RRs for responses

```
lsu.edu. 0 IN NSEC3PARAM 1 0 10 6F772A6B
```

- Zone Name
- alg#
- iters
- Salt
- flags
Authenticated negative answer (NSEC3 nxdomain)

(Example taken from RFC 5155 Appendix B. Consult for details)

Question: a.c.x.w.example. IN A

;; AUTHORITY SECTION:
0p9mhaveqvm6t7vb15lop2u3t2rp3tom.example. NSEC3 1 1 12 aabbccddd (2t7b4g4vsa5mi47k6lm5bv1a22bojr MX DNSKEY NS SOA NSEC3PARAM RRSIG )
0p9mhaveqvm6t7vb15lop2u3t2rp3tom.example. RRSIG NSEC3 7 2 3600 (20150420235959 20051021000000 40430 example. OSgWSm26B+CS+dDL8b5QrWr/dEWhtCsklwlKL IBHYH6blRxK9rC0bMJPwQ4mLIuw85H2EY762 BOCXJZMnpuwhpA== )

b4um86eghd6nea196smvml04ors995.example. NSEC3 1 1 12 aabbccddd (gjeq5e26p1bf1g8nklp59enfd789njgi MX RRSIG )
b4um86eghd6nea196smvml04ors995.example. RRSIG NSEC3 7 2 3600 (20150420235959 20051021000000 40430 example. ZkPG3M32lmoHM6pa3D6gZFGB/rhL//Bs3Ohm 5u4m/CUiwtablEVoAkkzd7S9590eiX4aLX3 p0v0TSTy1TsIzg== )

35mthgpgcu1qg68fab165klnskn3dpvl.example. NSEC3 1 1 12 aabbccddd (b4um86eghd6nea196smvml04ors995 NS DS RRSIG )
35mthgpgcu1qg68fab165klnskn3dpvl.example. RRSIG NSEC3 7 2 3600 (20150420235959 20051021000000 40430 example. g6jPUUpduAjkRljUsN8gB4UagAX0Nxy9shwQ Aynzo8EUWH+z6hEIBlUTPGj15eZl16VhQqgZ XtAIR3chwgW+SA== )

covers "next closer name"
matches closest encloser
covers wildcard at closest encloser
Authenticated negative answer (NSEC3 nodata)

NOERROR (nodata) responses can be authenticated with one signed NSEC record, which just reports all available RRTYPEs at that name (for qtype != DS)

In the example below blah.huque.com exists (TXT) but not for the MX record type.

$ dig +dnssec blah.huque.com. MX

;; -->>HEADER<<-- opcode: QUERY, status: NOERROR, id: 65366

;; AUTHORITY SECTION:

[SOA and RRSIG(SOA) omitted for brevity]

Q9T0VRM5S6EEF2N72RPCC5ENOFO4IGV3O.huque.com. 3284 IN RRSIG NSEC3 8 3 3600 ( 2012114122449 20121015121429 14703 huque.com.
1Su/WBJb3rBsU8ObV4bChPIMWcK93ac1B4b0Pq14m+Zo
X0kgu+PAqWLBm8FFeWwnT74XOWMXe+jvNMLSQ/nWfEjE
s+15lWsm4XJma0Pl+SoSHdIq1vJ9KfeEiWD8xLbpKH/N
3qwjnf4p4Fcm8LB6va4niZiJulMGNFzgRmtFDE = )

Q9T0VRM5S6EEF2N72RPCC5ENOFO4IGV3O.huque.com. 3284 IN NSEC3 1 0 5 9EBA4228 ( 1M2GGNB8TPS14SPF73V8RJS95FLHBNCO TXT RRSIG )

Hash of blah.huque.com.  
Next hashed name  
Type bitmap
NSEC3 with Opt-Out

• NSEC3 RRs exist only at signed delegations
  • Can prove existence of a secure delegation
  • Cannot prove non-existence of a name. Can only prove that delegation is insecure (because its hash falls between hashes of 2 secure delegations)

• Ideal for zones with relatively few secure delegations
  • Reduced zone size: fewer NSEC3 RRs and their signatures
  • This was important for signing very large zones like .COM
EDNS0

• DNS messages larger than 512 bytes requires:
  • Use of TCP (typically truncated UDP response followed by TCP retry)
  • EDNS0 - a DNS extension mechanism allowing negotiation of larger UDP message buffers
  • RFC 6891 “Extension Mechanisms for DNS (EDNS0)"

• For DNSSEC, EDNS0 does:
  • Negotiation of larger UDP payload sizes
  • Flag to indicate querier is able to process DNSSEC records: the “DNSSEC OK” or “DO” bit
Opt “pseudo” RR

• OPT resource record (RR type code 41)
• Pseudo RR (doesn’t exist as data in a zone)
• Appears in the “Additional Section” of a DNS message
• Contains maximum UDP Payload Size, extended RCODEs and flags
• Only flag defined to date: DNSSEC OK (DO)
New Header Flags: CD, AD

- AD - “Authenticated Data”
- CD - “Checking Disabled”
AD Flag

• AD - “Authenticated Data”

• Resolver sets this flag in responses when the queried record is signed with a valid, unexpired signature and an authenticated chain of trust all the way to a configured trust anchor (which could be the preconfigured/tracked root key)

• All data in the included answer and authority sections has been appropriately authenticated by the resolver

• Can also be set in a DNS query - to indicate querier understands responses with AD bit (eg. if it wants authenticated state but not necessarily DNSSEC RRs)
CD Flag

• CD - “Checking Disabled”

• Querier sets CD flag to indicate that “pending” (non-authenticated data) is acceptable to it, eg. because it is willing to do its own cryptographic validation of the signatures.

• DNSSEC enabled servers must not return “bad” data (eg. that have bad signatures) though (* is this true in practice?)

• Commonly used by 'validating stub resolvers'
DNS Packet Format

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS Header (12 bytes)</td>
</tr>
<tr>
<td>Question Section</td>
</tr>
<tr>
<td>Answer Section</td>
</tr>
<tr>
<td>Authority Section</td>
</tr>
<tr>
<td>Additional Section</td>
</tr>
</tbody>
</table>

- new AD, CD flags
- new DNSSEC RRs can appear here (DNSKEY, RRSIG, NSEC, NSEC3, etc)
- OPT RR with EDNS0 flags in the additional section, setting DO bit
## DNS Header

<table>
<thead>
<tr>
<th>QR</th>
<th>OpCode</th>
<th>AA</th>
<th>TC</th>
<th>RD</th>
<th>RA</th>
<th>Z</th>
<th>AD</th>
<th>CD</th>
<th>RCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12-bytes

- **16-bit Query ID**
- **QDCOUNT** (#records in query)
- **ANCOUNT** (#records in answer)
- **NSCOUNT** (#records in authority)
- **ARCOUNT** (#records in additional)
DNS Response Codes

Common Response codes:

0    NOERROR   No Error
1    FORMERR   Format Error
2    SERVFAIL  Server Failure
3    NXDOMAIN  Not existent domain name
4    NOTIMPL   Function not implemented
5    REFUSED   Query Refused, usually by policy

Standard response code for DNSSEC responses that fail to authenticate (validate) properly, e.g. bad signature, expired signature etc is SERVFAIL
Extended RCodes

Extended RCODES do not appear in the DNS header (since there isn’t enough space there). They instead appear in the OPT pseudo RR, which has a special format designed to accommodate them.

Extended RCodes used by EDNS0, TSIG, TKEY, etc:

- BADVERS: Bad OPT version (16)
- BADSIG: TSIG Signature Failure (16)
- BADKEY: Key not recognized (17)
- BADTIME: Signature out of time window (18)
- BADMODE: Bad TKEY Mode (19)
- BADNAME: Duplicate Key Name (20)
- BADALG: Algorithm not supported (21)
- BADTRUNK: Bad Truncation (22)
Other DNSSEC caveats
General DNSSEC Caveats

• Zone size increases significantly when signed

• Memory and CPU usage increase

• DNSSEC answers are larger

• Server side & query side impacts

• Interference by firewalls, proxies, and other middlebox, eg. botching EDNS0, large packets, DNSSEC meta data, not passing all UDP fragments, etc

• Fallback to TCP increases

• Many modern resolvers already ask for DNSSEC by default (ie. set the DNSSEC-OK bit in their queries)
Amplification Attacks

• Increased susceptibility to Distributed Denial of Service (DDoS) attacks, using DNS response amplification

• Look at Response Rate Limiting and other countermeasures
  • [http://www.redbarn.org/dns/ratelimits](http://www.redbarn.org/dns/ratelimits)
Other measures?

• Ideally, these shouldn’t be necessary but ...
• If needed, to workaround some types of firewalls and middleboxes (on at least one server)
• Constrain EDNS0 payload size (< PMTU)
  • eg. “edns-udp-size 1472”
• Configure minimal-responses (“minimal-responses yes”)
• Make sure DNS over TCP is allowed (see RFC 5966) - you should always do this!
Accurate time

• DNSSEC has an important dependency on accurate time
  • Validating resolvers need to check signature validity time
  • Signing servers need to produce correct signature validity intervals

• Make sure your servers have accurate time

• I’d recommend configuring them to get authenticated time from an NTP server
DNSSEC Deployment Status
DNSSEC Deployment status

• DNS Root was signed: July 2010
• .EDU, .NET, .COM followed in 2011/12
• Top Level Domain status as of Jan 3\textsuperscript{rd} 2020:
  • All TLD 1385 of 1515 signed (91.4%)
  • ccTLD: 176 of 304 signed (57.9%)
  • New gTLD: all are signed (1187) – contractual requirement for DNSSEC

• Details see: \url{http://stats.research.icann.org/dns/tld_report/}
DNSSEC Deployment status

• Levels beneath the TLDs are where a lot more needs to happen

• But good progress in certain pockets:
  • US Government agencies: ~ 82% (impetus: FISMA OMB mandate)
  • .NL (Netherlands): over 3 million signed delegations (~ 55%)
  • .BR (Brazil): over 700K signed delegations (~ 40%)

• Disappointing low adoption generally:
  • .COM: ~ 1.5 million signed delegations (~ 1% only)
  • .ORG and .NET are similar
DNSSEC Validation

- US GOV/FISMA IT security policy DNSSEC validation mandate (2014)

- Public Resolvers:
  - Google Public DNS (8.8.8.8 etc)
  - Cloudflare DNS (1.1.1.1)
  - Quad9 (9.9.9.9)
  - OpenDNS/Cisco (!), and others ..

- ISPs:
  - Comcast DNS

- Worldwide, there is quite substantial use of validating resolvers.
What percentage of DNS Resolvers in each country perform DNSSEC validation?

https://stats.labs.apnic.net/dnssec
DNSSEC common challenges

• CDNs and Dynamic services
  • Original pre-computed signature model was a challenge
  • Today, there are many online signing implementations though

• Key Rollovers and zone maintenance (better tooling?)

• Perception: no compelling need
  • Applications are ultimately protected anyway at higher layers, like with TLS and Certificates
  • But defense in depth, and the need to attack detects as early as possible, mean that all layers of the stack should be cryptographically protected, that includes like DNS (DNSSEC), and Routing (RPKI, SBGP etc).

• No ”Killer App” (but see “DANE”)
Registrar support

• Note: not all TLD registrars support DNSSEC yet (i.e. ability to install a DS record in the TLD)

• The situation is improving

• ICANN maintains a list at:
Gory details ...

- RFC 4033: DNSSEC Introduction
- RFC 4034: Resource Records for DNSSEC
- RFC 4035: DNSSEC - Protocol modifications
- RFC 5155: Hashed Authenticated Denial of Existence (NSEC3)
- RFC 6781: DNSSEC Operational Practices
- RFC 6840: Clarifications & Implementation Notes for DNSSEC
- (and a few other related ones ...)
Diagnostic Tools
Zone/validation testers

- Checking correct operation/deployment:
  - DNSviz: [http://dnsviz.net/](http://dnsviz.net/)
  - DNSCheck: [http://dnscheck.iis.se/](http://dnscheck.iis.se/)

- DNSSEC Validation testing
  - [http://dnssectest.sidn.nl/](http://dnssectest.sidn.nl/)
  - [http://test.dnssec-or-not.com/](http://test.dnssec-or-not.com/)
Zone maintenance

• 3rd party tools that some folks use to deploy/manage DNSSEC with BIND (mostly everything can be done in BIND itself these days):
  • OpenDNSSEC
  • zkt
• Microsoft DNSSEC deployment guide
DNSSEC zone and trust chain visualizer/debugger

http://dnsviz.net/
DNSSEC debugger

`dnssec-debugger.verisignlabs.com`
Last hop security?
Securing the last hop?

- **endstation** (uses DNS stub resolver)
- **recursive resolver** (has root’s pubkey)
- **www.upenn.edu**
  - set DO bit
- **www.upenn.edu**
  - referral to .edu + DS, RRSIG
  - referral to upenn.edu + DS, RRSIG
- **upenn.edu**
  - upenn pubkey
- **.edu**
  - edu pubkey
- **. (root)** root’s pubkey

**Stub to Recursive Resolver channel**

1. **www.upenn.edu**
2. **recursive resolver**
3. referral to .edu + DS, RRSIG
4. referral to upenn.edu + DS, RRSIG
5. upenn.edu
6. www.upenn.edu
7. answer 1.2.3.4 + RRSIG
8. **endstation**
Securing the last hop

• How do we protect the stub resolver?

• Employ a channel security mechanism between stub and the upstream recursive resolver:
  • TSIG, SIG(0), IPSEC, etc

• Have the stub validate DNSSEC responses? Set CD bit and authenticate signatures directly?

• Run a full service validating DNS Resolver on clients?

• DNS over TLS/HTTPS – see DNS Privacy Tutorial
Channel Security

• For stub channel security, simple symmetric key TSIG won’t work

• Can’t distribute same TSIG key to many clients, because that allows any of them to forge answers to all others

• Need per client keys and thus a key management infrastructure

• GSS-TSIG has a chicken-egg problem, because DNS is often used to locate Kerberos servers

• SIG(0) may be better - distribute single public key to clients

• Microsoft has an implementation of IPsec (GSS authenticated)
Key Rollover
Key Rollover

• Conventional wisdom is that DNSSEC keys should be changed ("rolled over") at regular intervals. However, not everyone agrees, including some noted security experts.

• If you choose strong enough keys, there is no cryptographic reason to routinely roll them.

• There are good operational reasons to change keys after specific events, eg. turnover of a staff member who had access to the private keys, or a system compromise of the server.

• Some argue routine key rollover instills practice & confidence that you’ll be able to do it properly when you really need to. However, do we do this for other applications (Kerberos, PKI/CAs, SSL)?
Key Rollover

• However, most sites do routinely change DNSSEC keys
  • In particular 1024-bit RSA keys are not strong enough for long term use, and are in fairly common use in DNSSEC for performance and packet size reasons
  
• Typically, ZSKs are rolled over more frequently (eg. a few times per year, this can be done transparently, and with no co-ordination with the parent zone)

• KSKs are rolled less frequently (typically once per year or less). This does require co-ordinating with the parent zone to sign and install new DS records for the KSKs.

• Note: ICANN is planning a rollover of the root KSK
Key Rollover

- RFC 6781: DNSSEC Operational Practices (v2)
  - Covers general practices, procedures, recommendations
- Most commonly used:
  - KSK rollover: double signature policy
  - ZSK rollover: pre-publish policy
KSK: Double signature

• Generate new KSK; publish (public part) in zone
• Sign DNSKEY RRset with both keys
• Publish additional DS record in parent for new key
• Wait until DS is propagated and TTL of the old DS record
• Remove the old KSK and re-sign DNSKEY RRset with only new key, and remove old DS record from parent
ZSK: Pre-publish

• Generate new ZSK, and publish the DNSKEY in the zone, but do not yet sign zone data with it

• Wait zone propagation time + TTL of the DNSKEY RRset

• Use new ZSK for signing zone records instead of old ZSK, but leave the old ZSK published in the zone

• Wait zone propagation time + largest TTL of all records in the zone

• Remove old key & re-sign DNSKEY RRset
Re-signing Records

• Regardless of key rollover, DNS records in a zone need to be re-signed periodically. Why?

• DNSSEC signatures have a validity period

• Limiting signature validity period reduces susceptibility to replay attacks in the event the data changes (i.e. ability for an attacker to replay a previously valid response)
Trust Anchor Updates

• RFC 5011: Automated Trust Anchor updates by resolvers
• A method to keep track of trust anchors (e.g. the root key) and automatically reconfigure resolvers as those trust anchors are updated (e.g. as a result of a scheduled key rollover)
Application use of DNSSEC
Application use of DNSSEC

• One of the more exciting prospects for DNSSEC

• DNSSEC can be employed to store cryptographic keys in the DNS

• DNSSEC allows applications to securely obtain (authenticate) those keys and use them in application security protocols

• Possible applications: SSH, SSL/TLS, HTTPS, S/MIME, PGP, SMTP, DKIM, and many others

• Existing records:
  • SSHFP, IPSECKEY, DKIM TXT
  • DANE records: TLSA, OPENPGPKEY, upcoming: SMIME, IPSECA
Application use of DNSSEC

• Securely obtaining other assertions from the DNS

• DKIM/ADSP

• Route Origination Authorizations (controversial - see RPKI, the standardized mechanism to do this, which will allow BGP path validation also)
SSHFP record

- SSH Host Key Fingerprint (RFC 4255)
- Allows you to validate SSH host keys using DNSSEC

In OpenSSH, you can use the client configuration directive "VerifyHostKeyDNS" to use this. Enabled by default in some newer operating systems like FreeBSD 10.
# SSHFP record

<table>
<thead>
<tr>
<th>value</th>
<th>public key algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RSA</td>
</tr>
<tr>
<td>2</td>
<td>DSA</td>
</tr>
<tr>
<td>3</td>
<td>ECDSA</td>
</tr>
<tr>
<td>4</td>
<td>Ed25519</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>value</th>
<th>fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SHA-1</td>
</tr>
<tr>
<td>2</td>
<td>SHA-256</td>
</tr>
</tbody>
</table>
IPSECKEY record

- RFC 4025: method for storing IPSEC keying material in DNS
- rdata format: precedence, gateway-type, algorithm, gateway address, public key (base64 encoded)
- Not much uptake; will likely be superseded by newer proposals like IPSECA

38.2.0.192.in-addr.arpa. 7200 IN IPSECKEY ( 10 1 2
192.0.2.38
AQNRU3mG7VT02BkR47usntb102uFJtugbo6BSGvgqt4AQ== )
TLS and the Internet PKI

• A very large number of security protocols authenticate server names (and sometimes clients) with X.509 certificates
  • TLS, IPsec, HTTPS, SIPS, SMTP, IMAP, XMPP, etc.

• These certificates are issued and signed by the Internet PKI, composed of a set of globally trusted public Certification Authorities (CAs)
Public CA model problems

• Applications need to trust a large number of global certificate authorities, and this trust appears to be unfounded

• No namespace constraints! Any of them can issue certificates for any entity on the Internet, whether you have a business relationship with them or not

• Least common denominator security: our collective security is equivalent to weakest one

• Furthermore, many of them issue subordinate CA certificates to their customers, again with no naming constraints

• Most CAs aren't capable of issuing certs with any but the most basic capabilities (eg. alternate name forms or other extensions)
Public CA model problems

• Analysis of the HTTPS Certificate Ecosystem:

• Approximately 1,800 separate entities are capable of issuing certificates for anyone!
DANE/TLSA record

• RFC 6698: The DNS-Based Authentication of Named Entities (DANE) Protocol for Transport Layer Security (TLS)
  • http://tools.ietf.org/html/rfc6698

• Use DNSSEC for better & more secure ways to authenticate SSL/TLS certificates:
  • by specifying authorized public CAs, allowable end entity certs, authorizing new non-public CAs, or even directly authenticating certs without involving CAs!

• New record type: TLSA
TLSA record example

_port, transport proto & server domain name_

_TLSA rrtype_

__443._tcp.www.example.com. IN TLSA ( 0 0 1 d2abde240d7cd3ee6b4b28c54df034b9 7983a1d16e8a410e4561cb106618e971 )

_usage_

_selector_

_matching type_

_certificate association data_
TLSCA rdata parameters

Usage field:
0  PKIX-TA: CA Constraint
1  PKIX-EE: Service Certificate Constraint
2  DANE-TA: Trust Anchor Assertion
3  DANE-EE: Domain Issued Certificate

Selector field:
0  Match full certificate
1  Match only SubjectPublicKeyInfo

Matching type field:
0  Exact match on selected content
1  SHA-256 hash of selected content
2  SHA-512 hash of selected content

Certificate Association Data: raw cert data in hex
## TLSA rdata parameters

**Usage field:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PKIX-TA: CA Constraint</td>
</tr>
<tr>
<td>1</td>
<td>PKIX-EE: Service Certificate Constraint</td>
</tr>
<tr>
<td>2</td>
<td>DANE-TA: Trust Anchor Assertion</td>
</tr>
<tr>
<td>3</td>
<td>DANE-EE: Domain Issued Certificate</td>
</tr>
</tbody>
</table>

**Selector field:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Match full certificate</td>
</tr>
<tr>
<td>1</td>
<td>Match only SubjectPublicKeyInfo</td>
</tr>
</tbody>
</table>

**Matching type field:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Exact match on selected content</td>
</tr>
<tr>
<td>1</td>
<td>SHA-256 hash of selected content</td>
</tr>
<tr>
<td>2</td>
<td>SHA-512 hash of selected content</td>
</tr>
</tbody>
</table>

Certificate Association Data: raw cert data in hex
TLSA usage types

0  PKIX-TA: CA Constraint
   Specify which CA should be trusted to authenticate the certificate for the service. Full PKIX certificate chain validation needs to be performed.

1  PKIX-EE: Service Certificate Constraint
   Define which specific service certificate ("EE cert") should be trusted for the service. Full PKIX cert validation needs to be performed.

2  DANE-TA: Trust Anchor Assertion
   Specify a domain operated CA which should be trusted independently to vouch for the service certificate.

3  DANE-EE: Domain Issued Certificate
   Define a specific service certificate for the service at this domain name.
TLSA record example

Usage type 1: Service certificate constraint; match an end-entity certificate

_443._tcp.www.example.com. IN TLSA ( 
  1 1 2 92003ba34942dc74152e2f2c408d29ec 
    a5a520e7f2e06bb944f4dca346baf63c 
    1b177615d466f6c4b71c216a50292bd5 
    8c9ebdd2f74e38fe51ffd48c43326cbc )
TLSA record example

(my own website; full cert assoc, no CA required)


;; ANSWER SECTION:
_443._tcp.www.huque.com. 7200 IN TLSA 3 0 1 ( 7EF4BD014E9A4F302FC1EE74FB2D29718C5B0F4CB23B 25B267A1D92F0410890B )

_443._tcp.www.huque.com. 7200 IN RRSIG TLSA 8 5 7200 ( 20131028121743 20130928111915 14703 huque.com. rjF6V1stQ050zG08s8m8DfBfqDvjqqzW3Im0Jc04HEDG fyvzQ1CDX7Dxnbk7ZBofFGtNsVlx5XGS57k0ZLURsRWt wY+pqzcJ1ELVo16iOwNs0v+h9ZDyCa1GF7gL4k3DyKVe 6cLquFa7RlywORqLYF32+adUP88/j63MmehR2VA= )
TLSA record (SMTP e.g.)

$ dig +dnssec +multi _25._tcp.nlnetlabs.nl. TLSA

;; ANSWER SECTION:
_25._tcp.nlnetlabs.nl. 10200 IN CNAME 3.1.1._dane.nlnetlabs.nl.
3.1.1._dane.nlnetlabs.nl. 10200 IN RRSIG CNAME 8 4 10200 ( 20130529005004 20130501005004 42393 nlnetlabs.nl. SNKS6Bo8SsqRxDuxF9dRiwqom4YqOArpLAWjv1WHf5fr aURdyssZ3V/R8jBRwMNhQNqIQV1Dc4i84OsBs2Vpolil j0Gy5mfqgnxRCh5b6TtLDE5t41cFg0k5FgaqtLXCd0an f8zdv8nQM/9U0aXgnQLXuUDv4ZpDPXkxPuokKIE= )
3.1.1._dane.nlnetlabs.nl. 10200 IN TLSA 3 1 1 ( 0D1FCBD71686199607A132744A4918FC209565C91FA8 E9FFEEA0AAFD6B9305F6 )
3.1.1._dane.nlnetlabs.nl. 10200 IN RRSIG TLSA 8 6 10200 ( 20130529005004 20130501005004 42393 nlnetlabs.nl. mE8cSI5wCbx41sQTHoWZTweh1Jo+A0ZDEtNNDGKJvafL 2Q7cMhoqq9J5mvaKFm1MN8qqiaRbt56c90cahFA3xkO3 l0D1jUlcU1xPVoRDzWe73MjjuU76UrsyqNdxmHKB6xR mEFxkvcQ5EM6b1fDGRH0fnMFV15ezi9GwkB7DcI= )
DANE/TLSA tools

• TLSA record generation and testing:
  • Command line tools: swede, hash-slinger, ldns-dane, ...
  • Web based tool: https://www.huque.com/bin/gen_tlsa
  • Testing: https://dane.sys4.de/

• TLSA validators:
  • Some 3rd party validator plugins (Firefox, Chrome, Opera, Safari)
    • https://www.dnssec-validator.cz/
  • Bloodhound Mozilla fork https://www.dnssec-tools.org/download/
  • http://www.internetsociety.org/deploy360/resources/dane/
The End.