13

SECURING DNS (PART II): DNSSEC

When I sign a letter or check, I inherently demonstrate my approval and authorization by virtue of my signature.* When I sign more important documents, such as a mortgage note, I need to have my signature validated, typically through a notary public. The notary verifies my identity and also validates my signature generally by comparing it with a driver's license or passport signature. By stamping my mortgage note, the notary confirms that it is I that signed the document and therefore my signature is trusted. DNSSEC works in a loosely analogous fashion. A resolver, or recursive server resolving on behalf of a stub resolver receives resolution data along with a signature on the data. As long as I trust the signer, I can validate the data using the signature. The element of trust requires some initial configuration of trust information from the signer in the form of trusted keys, which are used to verify the trustworthiness of the signer of the data received. If I don't trust the signer directly, I need to seek signature validation by seeking an entity that I trust that will "vouch for" the signer. Not that my mortgage company doesn't trust me, but they required validation of my signature!

DNS security extensions, DNSSEC, originally defined in RFC 2535 (140), was modified and recast as DNSSEC*bis*, defined in RFCs 4033–4035 (114, 141, 142). This

* This basic introduction is from Chapter 9 of Ref. 11 and the high level description therein is expanded in more detail in this chapter.

IP Address Management: Principles and Practice, by Timothy Rooney

recasting was due to scalability issues with the original specification. While still being tweaked to some degree, DNSSEC*bis*, hereafter referred to simply as DNSSEC, provides a means to authenticate the origin of resolution data within DNS and to verify the integrity of that data. DNSSEC also provides a means to authenticate the nonexistence of DNS data, allowing the signature of "not found" resolutions (e.g., NXDOMAIN) as well. Thus, DNSSEC enables detection of packet interception, ID guessing, and cache poisoning attacks on both resolution data and on "not found" resolutions. DNSSEC provides these services through the use of asymmetric public key cryptography technology to perform data origin authentication and end-to-end data integrity verification.

13.1 DIGITAL SIGNATURES

We introduced the concept and process for digital signature generation and verification in Chapter 10 within the context of DKIM, but we'll review it briefly here for convenience. Digital signatures enable the originator of a given set of data to sign the data using a private key such that those receiving the data and the signature, along with a corresponding public key for deciphering the signature, can perform data origin and integrity verification. DNSSEC uses an asymmetric key pair (private key/public key) model. In such a model, data signed with a private key can be validated by deciphering the data with the corresponding public key. The private key and public key form a key pair. Conceptually, the private/public key pairs provide a means for holders of the public key to verify that data was signed using the corresponding private key. This provides authentication that the data verified was indeed signed by the holder of the private key. Digital signatures also enable verification that the data received matches the data published and was not tampered with in transit.

Refer to Figure 13.1. The data originator, shown on the left of the figure, generates a private key/public key pair and utilizes the private key to sign the data. The first step in signing the data is to produce a hash of the data, sometimes also referred to as a digest. Hashes are one-way functions[†] that scramble data into a fixed length string for simpler

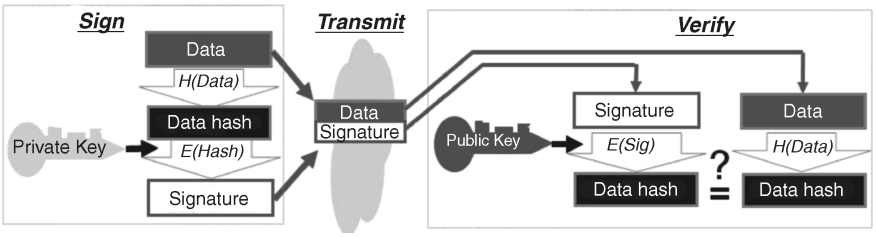

Figure 13.1. Digital signature creation and verification process (11).

[†] A one-way function means that the original data is not uniquely derivable from the hash. That is, one can apply an algorithm to create the hash, but there is no inverse algorithm to perform on the hash to arrive at the original data.

manipulation, and represent a "fingerprint" of the data. This means that it is very unlikely that another data input could produce the same hash value. Thus, hashes are often used as checksums but don't provide any origin authentication (anyone knowing the hash algorithm can simply hash arbitrary data). Common hash algorithms include HMAC-MD5, RSA-SHA-1, and RSA-SHA-256. The hash is encrypted using the private key to produce the signature. The encryption algorithm is fed the hash and the private key to produce the signature.

Both the data and its associated signature are transmitted to the recipient. Note that the data itself is not encrypted, merely signed. The recipient must have access to the public key that corresponds with the private key used to sign the data. In some cases a secure (trusted) public key distribution system such as a public key infrastructure (PKI) is used to make public keys available. In the case of DNSSEC, public keys are published within DNS, along with the resolution information and corresponding signature.

The recipient computes a hash of the received data, as did the data originator. The recipient applies the encryption algorithm to the received signature using the originator's public key. This operation is the inverse of the signature production process and produces the original data hash as its output. The output of this decryption, the original data hash, is compared with the recipient's computed hash of the data. If they match, the data has not been modified and the private key holder signed the data. If the private key holder can be trusted, the data can be considered validated.

13.2 DNSSEC OVERVIEW

DNSSEC utilizes this asymmetric key pair cryptographic approach to provide data origin authentication and end-to-end data integrity assurance. Any attempt to spoof or otherwise modify data en route to the destination will be detected by the recipient, that is, the resolver or more typically, its recursive/caching DNS server on its behalf. This feature makes DNSSEC an effective mitigation strategy against man-in-the-middle and cache poisoning attacks.

The original DNSSEC*bis* specifications do not account for a secure key distribution system, so one or more trusted keys have to be manually configured on the resolver or recursive name server.[‡] However, a subsequent specification, RFC 5011 (141), defines a means to ease this process by authenticating new and revoked trusted keys based on a manually configured initial key. This initial key serves as the "initial condition" in rolling forward over time with new, revoked, and deleted keys. We'll talk about this automated trusted key update process a bit later. Whether configured manually or updated automatically, each trusted key identifies the public key corresponding to a given trusted zone as authorized by the zone administrator.

This is analogous to the bank notary being trusted by the bank to validate my identity. After all, any imposter may sign invalid zone data with a private key and publish the

[‡] Pragmatically, the term "resolver" in the context of DNSSEC refers to the resolver function of the recursive server, which resolves the queried information and verifies signatures as well. Considering our deployment example from Chapter 11, the Internet Caching servers would perform this signature validation function.

corresponding data, signatures, and public key. Thus, the recursive server must be configured *a priori* with a key or set of keys that are trusted corresponding to trusted signed zones. The current public key in use by the trusted zone administrator must be conveyed to the resolver administrator out of band, or using a mechanism other than DNS. With the automed key update process just mentioned, an initial key must be configured for each trusted zone; however, ongoing key updates are performed using the DNS protocol.

A given trusted zone can authenticate a child zone's public key, extending the trust model from just the trusted zone to the trusted zone and its authenticated child zones. Likewise, these child zones can authenticate their children and so on, forming a *chain of trust* from the trusted zone to all signed delegated zones. With the Internet root zone now signed and major TLDs signed or soon to be signed, the chain of trust will emanate from the root trust anchor to TLDs, to lower level signed zones down the domain tree.

Configuration of trusted keys requires creation of a trust relationship with a zone administrator to obtain his/her public key. With a signed root and TLDs, this simplifies the trust model, requiring trust of the root zone and configuration of the root zone trust anchor. In lieu of (really as a predecessor to) using root zone keys, ISC has also created a trusted key registry (dlv.isc.org) as a repository of trusted keys for registered domains to enable "lookaside" validation in acting as a "parent zone proxy," reducing the requirement impact of forming individual relationships with each domain administrator.

While not explicitly required by DNSSEC specifications, operational experience has led to the recommendation that two keys be used per zone a zone signing key (ZSK) and a key signing key (KSK).[†] As we will see later, this streamlines the complex key rollover process while trading off key length security and complexity against nimbly changing zone signing keys as needed. At this point, suffice it to say that the ZSK is used to sign the data within the zone and the KSK is a longer term key that signs the ZSK. Both the ZSK and KSK are each comprised of a public and private key pair. The private keys are used to sign zone information and must be secured, ideally on a secure server or host. The corresponding public keys are published within the zone file in the form of DNSKEY resource records.

The public KSK of the trusted zone is the trusted key[‡] configured in each recursive server, which should match the corresponding KSK DNSKEY resource record published in the respective zone file. The resolved data's signature is validated using the zone's ZSK, and the ZSK is signed by and its signature is thereby validated by the trusted KSK.

If this KSK is not trusted, an attempt is made to check if the parent zone is signed or if lookaside validation is configured. If signed, this parent zone (or lookaside registry) signs its delegation to the child by signing the child's public KSK in the form of a delegation signer (DS) record (or DNSSEC Lookaside Validation, DLV record) in the parent zone. This delegation in turn is signed with the parent's ZSK which itself is signed with the

[†] The motivation for this recommendation and discussion of other DNSSEC operational practices are discussed in RFC 4641 (167).

[‡] A trusted key is synonymous with a trust anchor, which is also known as a Secure Entry Point (SEP) into the DNS domain tree.

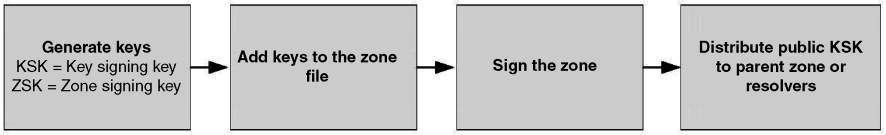

Figure 13.2. Basic DNSSEC implementation steps (11).

parent's KSK. Once again, if these signatures are valid and the KSK matches a trusted key, the resolution is complete and secure. Otherwise, the process continues with the parent's parent zone and so on.

The validation process works up the chain of trust to a matching trusted key, which if found, deems the resolution data validated. Otherwise, it will be considered insecure.

13.3 CONFIGURING DNSSEC

The process of implementing DNSSEC involves creating private/public key pairs, adding the public key information to the zone file to be signed, signing the zone with the corresponding private keys, and distributing the public KSK information to either parent zone administrators or to resolver administrators who trust you and your zone information. Figure 13.2 illustrates the basic process.

Let's now illustrate this basic process looking at the mechanics for implementing DNSSEC. We'll illustrate the process using manual and automated* BIND methods and utilities (144), which support DNSSEC*bis* today. Microsoft supports DNSSEC*bis* in its Windows Server 2008 R2 release. We'll demonstrate the implementation of DNSSEC by signing the ipamworldwide.com zone file after we review these steps in more detail.

13.3.1 Generate Keys

Our first step is to generate keys that will be used to sign our zone information. BIND ships with the `dnssec-keygen` utility which provides a simple command line to generate a private/public key pair. It even creates the DNSKEY record! To create a ZSK key pair for our ipamworldwide.com zone, we use the `dnssec-keygen` command:

```
dnssec-keygen –a RSA-SHA-1–b 1024–n ZONE –c IN –e ipamworldwide.com
```

This utility is used not only to create DNSSEC keys, but can also be used to create TSIG keys and KEY records. The arguments, all of which are optional unless otherwise specified, within the `dnssec-keygen` utility include

- `-a` *`algorithm`*: (required) where algorithm for DNSSEC keys may be
  - RSA-SHA-1

* BIND 9.7.0 introduced several new key and signature management features to automate many of these steps as we will describe.

- RSA-SHA-256
- RSA-SHA-512
- NSEC3-RSA-SHA-1 (RSA-SHA-1 algorithm with a signal that the zone signed with this key may use NSEC3)
- DSA (Digital Signature Algorithm)
- NSEC3DSA (DSA algorithm with a signal that the zone signed with this key may use NSEC3)
- RSA-MD5

- `-b keysize`: (required) specifies the number of bits in the key. Valid keysizes for each algorithm are
  - RSA-SHA keys 512–2048 bits
  - DSA keys. 512–1024 bits, divisible by 64
- `-n nametype`: (required) identifies the type of key owner. Valid values include ZONE (default, for DNSKEY), HOST, ENTITY, USER, or OTHER.
- `-3`: use an NSEC3-capable key generating algorithm (NSEC3-RSA-SHA-1 is used if no `–a` argument is specified). RSA-SHA-256 and RSA-SHA-512 are also NSEC3-capable.
- `-A date/offset`: sets the activation date of the key. The date/offset field is an absolute date/time when expressed in either YYYYMMDD or YYYYMMDDHHMMSS format (or `none` to unset) or as an offset from the present time when a "+" or "−" prefix is used with either of these date formats. When not set, and `–G` is not set, the default is "now."
- `-C`: option to generate the private key without any metadata relating to creation, publication, and/or activation dates, which may be incompatible with older BIND versions.
- `-c class`: class of the DNS resource record containing the key.
- `-D date/offset`: defines the date or offset from the present time when this key is to be deleted from the zone. The date/offset field is an absolute date/time when expressed in either YYYYMMDD or YYYYMMDDHHMMSS format (or `none` to unset) or as an offset from the present time when a "+" or "−" prefix is used with either of these date formats. At the specfied time, the key will be removed from the zone, though it will remain in the key repository.
- `-e`: command option to use a large exponent when using the RSA-MD5 or RSA-SHA-1 algorithms.
- `-E engine`: command option to use crypto hardware (OpenSSL engine) for random number generation as well as key generation when supported. The default is `pkcs11` when compiled with PKCS #11 support or `none` otherwise.
- `-f flag`: sets the Flags field in the DNSKEY (or KEY) resource record; currently, *flag* = KSK is used to create a KSK (sets the SEP bit in the DNSKEY record); flag = REVOKE sets the Revoke flag for this key.

- -g *generator*: specifies a key generator value for the DH algorithm.
- -G: generates a key which is not to be published or used for signing.
- -h: prints a help summary for this command.
- -I *date/offset*: sets the date/offset when the key is to be retired. The date/offset field is an absolute date/time when expressed in either YYYYMMDD or YYYYMMDDHHMMSS format (or none to unset) or as an offset from the present time when a "+" or "−" prefix is used with either of these date formats. When retired, the key remains in the zone but is no longer used to sign the zone.
- -k: indicates that a KEY record is to be created, not DNSKEY; deprecated in favor of the –T option.
- -K *directory*: defines the directory in which the key files will be placed.
- -p *protocol*: sets the Protocol field in the resource record. For DNSSEC, the default value of 3 is used.
- -P *date/offset*: sets the date/offset when the key is to be published in the zone file, though not used to sign the zone. The date/offset field is an absolute date/time when expressed in either YYYYMMDD or YYYYMMDDHHMMSS format (or none to unset) or as an offset from the present time when a "+" or "−" prefix is used with either of these date formats.
- -q: quiet mode, which suppresses output, including indication of progress.
- -r *randomsource*: indicates a source or random data such as a file or character device such as the keyboard.
- -R *date/offset*: defines the date when the key is to be revoked. The date/offset field is an absolute date/time when expressed in either YYYYMMDD or YYYYMMDDHHMMSS format (or none to unset) or as an offset from the present time when a "+" or "−" prefix is used with either of these date formats. When revoked, the "revoke" bit is set in the corresponding DNSKEY resource record, though it will remain in the zone and be used to sign the zone.
- -s *strength*: specifies the strength value of the key, though not relevant to DNSSEC.
- -t *type*: indicates the use of the key to authenticate data (AUTH) and/or to encrypt data (CONF). AUTHCONF supports both functions, while NOAUTH, NOCONF, and NOAUTHCONF negate respective functions.
- -T *rrtype*: specifies an RRType to use for public key creation in resource record format. Valid values of *rrtype* include DNSKEY (default) or KEY.
- -v *level*: sets the debug level.
- *keyname*: (required) the name of the key, generally the zone name, which serves as the owner field of the DNSKEY record.

The five date/offset based options are new in BIND 9.7.0 and provide timing meta data for the key being generated and hence when it is to be used in the zone signing process. Thus, keys can be staged and rolled through their entire lifecycle, consisting of

publication in the zone, use for signing zone information, revocation, retirement, and deletion. To summarize these options and their use through a key's lifecycle

- -P. Defines the time when the key being generated is to be published in the zone file, though not used in signing.
- -A. Defines the activation time, when this key being generated is to be used to sign zone data. If this is a KSK and the update-check-ksk option is set to yes, this key will sign the DNSKEY RRSet only. Otherwise, it will be used to sign all zone RRSets, allowing for cases where a single zone key is implemented.
- -R. Defines the time when this key is to be revoked. This defines the date when the revoke flag will be set in the corresponding DNSKEY record. This key (with the revoke bit set) will still be used to sign zone data but resolvers will be on notice that this key is revoked.
- -I. Defines the retire time for the key, after which this key will not be used to sign zone data, though the key will remain in the zone file.
- -D. Defines the time when the key will be deleted from the zone file.

These timing options enable you to define the entire key lifecycle at key generation time!

An alternative key file generation utility first included with the BIND 9.6.0 distribution allows use of the PKCS#11* API to interface to a cryptographic token generating hardware device. The `dnssec-keyfromlabel` utility gets keys from the cryptographic hardware device and generates the public and private key files. This utility accepts the following arguments in exactly the same format as `dnssec-keygen` plus a new required parameter, indicating the key label.

- `-a` *`algorithm`*: (required) same values as `dnssec-keygen`
- `-3`: same meaning as `dnssec-keygen`
- `-c` *`class`*: same values as `dnssec-keygen`
- `-C`–: same meaning as `dnssec-keygen`
- `-E` *`engine`*: same values as `dnssec-keygen`
- `-f` *`flag`*: same values as `dnssec-keygen`
- `-G`: same meaning as `dnssec-keygen`
- `-C`–: same meaning as `dnssec-keygen`
- `-h`: same meaning as `dnssec-keygen`
- `-k`: same meaning as `dnssec-keygen`
- `-K` *`directory`*: same meaning as `dnssec-keygen`
- `-l label`: (required) label for keys on the PKCS#11 device
- `-n` *`nametype`*: same values as `dnssec-keygen`

*PKCS#11 is among the family of Public-Key Cryptography Standards published by RSA Laboratories.

- -p *protocol*: same values as dnssec-keygen
- -t *type*: same values as dnssec-keygen
- -v *level*: same values as dnssec-keygen
- -y: allows creation of DNSSEC key files even if the key ID collides with an existing key
- *keyname*: (required)

The meta data timing options discussed above are also supported. Both dnssec-keygen and dnssec-keyfromlabel return the key name. In our example, the result was

Kipamworldwide.com. + 005 + 14522

The format of the key name follows this convention

- K (for key)
- Keyname (ipamworldwide.com.)
- Key creation algorithm (005 = RSA-SHA-1 in this case)
- Key tag or identity of the key (14522)

The key tag provides a convenient way to refer to keys as we'll see a bit later. Two files were created by dnssec-keygen or dnssec-keyfromlabel using this name one with extension .private indicating the private key and the other with extension .key, containing the public key in the form of a DNSKEY record. In our example, the two key files were named

```
Kipamworldwide.com.+005+14522.private
Kipamworldwide.com.+005+14522.key
```

The Kipamworldwide.com. + 005 + 14522.private file contains the private key details including the format, algorithm, modules, exponents, primes, and coefficient values such as that shown below for the output of the dnssec-keygen command (blank lines inserted for improved readability).

```
Private-key-format: v1.2

Algorithm: 5 (RSASHA1)

Modulus:
x6QAwJiz6hHa/eUI2pGz6rvwEYpJdi1TJH8Uj4lDPTmzseCOgFEqB3/dZB0Q
  5LEs1ZetAJJEk4F+WccRKwqnIcGkvIKfTC8hn+gbiBAnadQRFLxNMBs6KB0e+
  yqiNK60sbrn22F8AYRiG3n2rTQndVtkaZep9jbcCqfu/DagB10=

PublicExponent: AQAAAAE=

PrivateExponent:
CWheqbbkIx3kRIa7NyDbdwZYGA83uBtdfnBTu8QyV8/h419T3fyWrWfKo4wi
  Vys9ql0Xmumwy/hSLmZJJrzxS6SVwaM/iEunsyyiHedeVKiMeYVlOlvJ3+
  OweKy/59y3drJS+qAm+cbtrhWZheXtzgR78wp2IK+4kHAhZTCYGAE=
```

```
Prime1:
8YuU4sicmKmu5Cz4IUjvE2kQit5pJPV3yUK04nPz9P0MJFKyCIAdsw2A5HoRn3++
  I5BtDjeQxkD0aFGA4S0fKXQ==
Prime2:
05ZzyiaiZK1JqQMCgT977NZkEuKgXI4seTUL1Wu7Z/FRs/7xHE4oSJrx7siwLOx
  WJKcc4Fo+4erVRHioiOadhAQ==
Exponent1:
Hpy1z37UsfdONCV7Kd/8xu07PslhtbX7EFVGRno/dOrWNp5p64hVhF5tbnNBVz
  ZHRQ+5IZzwMfQ3A3+GjY8QQQ==
Exponent2:
jfw+s9zt8uVMwubwowwxOsjX32GO3VrSPk68+CisiAVxYS8EdTOqvpYps6Vz+
  rJNnnk45urnlqDbWCx2tugyAQ==
Coefficient:
uUC/aKgEvOQymCmMukC4ExTm/7ly2w31V/NMOF2GzC7fc1gYvDZEOX6YNnz5e8
  PRD2bQXCTgsMorRs7PJYI2Cg==
```

The `Kipamworldwide.com.+005+14522.key` file is a bit easier to digest, containing our DNSKEY resource record

```
ipamworldwide.com. IN DNSKEY 256 3 5
      BQEAAAABx6QAwJiz6hHa/eUI2pGz6rvwEYpJdi1TJH8Uj4lDPTmzseCO
      gFEqB3/dZB0Q5LEs1ZetAJJEk4F+WccRKwqnIcGkvIKfTC8hn+gbiBAn
      adQRFLxNMBs6KB0e+yqiNK60sbrn22F8AYRiG3n2rTQndVtkaZep9jbc
        Cqfu/DagB10=
```

The interpretation or format of the DNSKEY resource record is depicted below.

Owner	TTL	Class	Type	RData			
Zone name	TTL	IN	DNSKEY	Flags	Proto	Alg	Key
ipamworldwide.com.	86400	IN	DNSKEY	256	3	5	BQEAAA...

The owner field defines the zone name. The RData consists of the following subfields

- *Flags*. Indicates that this key is a zone key (value = 256). Currently defined values for the Flags field are as follows. Using the decimal values below, we can see that a ZSK will have a flags value of 256, while a KSK will have an odd value, likely 257.
  - Bit 7. ZSK (Decimal = 256)
  - Bit 8. Revoke Signature (Decimal = 128)
  - Bit 15. KSK or secure entry point (SEP) (Decimal = 1)
  - Other bits. Unassigned
- *Protocol*. Must have a value of "3" indicating DNSSEC (this is the only value currently defined).

- *Algorithm*. Defines the algorithm used in key generation. Algorithms currently supported are encoded as follows
  - Value = 1. RSA-MD5, which is not recommended according to RFC 4034.
  - Value = 2. Diffie–Hellman
  - Value = 3. DSA-SHA-1
  - Value = 4. Reserved for Elliptic Curve
  - Value = 5. RSA-SHA-1, which is mandatory according to RFC 4034.
  - Value = 6. DSA-NSEC3-SHA1
  - Value = 7. RSASHA1-NSEC3-SHA1
  - Value = 8. RSA-SHA-256
  - Value = 10. RSA-SHA-512
  - Value = 12. GOST R 34.10-2001
  - Value = 252. Indirect
  - Values 253–254. Private
  - Values = 0, 255. Reserved
  - Other values. Unassigned
- *Key*. The public key (ZSK or KSK).

We can now repeat the dnssec-keygen command, this time using the `–f KSK` argument, along with a longer key size, to create our KSK pair.

```
dnssec-keygen –a RSASHA1 –b 2048 –n ZONE –c IN –e –f KSK ipamworldwide.
com.
```

The command line response to this command is the key pair name, `Kipamworld-wide.com.+005+06082`.

The resulting DNSKEY record was created

```
ipamworldwide.com. IN DNSKEY 257 3 5
     AwEAAdSAwGoUBhtjpE8GLGN4ryt8yEq71DqdE+ij3boe9lmvpM02YZ1/
     AQxoHbyA7NqRr+8dsTM8OrF2yFRbcP1y0/9q37T0PqxL5HjAZ8HrDoW9
     R/pC3XyRe9pMzRNr4as+c/xEISfhxzvR84CndF5XvFeh3H0kVDeTb+7Q
     RrG7hnph4P8w4SMg76tBvxHLFmj3OdP8vIUpRAnexEAdclamj1ZSPjLc
     dICzpDvQB/LLsYxx8wx2h0vTvhxZklqmy1dPBtIZu2A551VIrU0xgCJx
     DjJGCgBbrp1C01tYSdqlA1I2HCL8eV7io/CxnCuSThPlXaPLySojJpXU
     gDomWgVYeo0=
```

Notice the Flags field value is 257 for the KSK versus 256 for the ZSK due to the setting of the SEP flag. We'll refer to the KSK by its keyid = 06082 (from the generated key name) and the ZSK by its keyid = 14522.

13.3.2 Add Keys to the Zone File

Before we sign the zone, we need to include our two DNSKEY resource records within the zone file. Since the key files contain our DNSKEY resource records, you can either cut from the file and paste into the zone or simply use a $INCLUDE statement for each file

```
$INCLUDE Kipamworldwide.com.+005+14522.key
$INCLUDE Kipamworldwide.com.+005+06082.key
```

Don't forget to increment your serial number too. It's a good idea to run `named-checkzone` first, before signing the zone with the `dnssec-signzone` utility.

13.3.3 Sign the Zone

The zone signature process utilizes another BIND utility, `dnssec-signzone`, which performs a number of functions to sign the zone. First, it canonically orders the resource records within the zone. This essentially alphabetizes the resource records within the zone. This facilitates grouping of resource records with common owner name, class, and type into resource record sets (RRSets) for signature application. The other reason for canonically ordering resource records is to identify gaps between RRSets within the zone file and population with Next SECure resource records, which provide authenticated denial of existence of a given resource record within a zone. An NSEC3PARAM resource record must be present in the zone file to generate NSEC3 records during the zone signature process.

After canonical ordering and insertion of NSEC[3] records, `dnssec-signzone` signs the RRSets within the zone file, including DNSKEY RRSets (previously $INCLUDE'd in our example) and NSEC[3] RRSets. The signed zone file contains the original RRSets, canonically ordered and signed with resource record signature (RRSIG) records. The file also includes an NSEC[3] record and its corresponding RRSIG record for each RRSet within the file. The only records not signed within the zone file are NS records for child zones. The child zone is authoritative for this information, not the parent; therefore, the parent does not authenticate their accuracy.

Fortunately the `dnssec-signzone` utility performs all of these steps automatically canonical ordering, NSEC[3] insertion, and RRSIG creation and insertion to render a signed zone. Here's the dnssec-signzone command we'll use to sign the ipamworldwide.com. zone.

```
dnssec-signzone -k Kipamworldwide.com.+005+06082 -l dlv-registry.
net -g -o ipamworldwide.com. -t db.ipamworldwide.com Kipamworld-
wide.com.+005+14522.key
```

The arguments within the dnssec-signzone utility include

- `-3 salt`: generate an NSEC3 chain when signing this zone using the specified *`salt`* value. The salt is specified in hex and a dash (`-3 -`) indicates that no salt should be used when generating the NSEC3 chain.
- `-a`: verify all generated signatures.

- -A: set the OPTOUT flag on all NSEC3 records when generating an NSEC3 chain, and do not generate NSEC3 records for unsigned child zones (insecure delegations).
- -c *class*: class of the DNS zone.
- -C: compatibility mode with older versions of dnssec-signzone; generates –*zonename* keyset in addition to dsset-*zonename* upon signing the *zonename* zone.
- -d *directory*: look in the specified directory for the dsset or keyset files to sign the zone.
- -e *end_time*: specifies the date and time when the generated resource record set signature records expire. The end_time may be specified relative to the current time using $+N$, where N is the number of seconds from the current time, or in absolute time using the format YYYYMMDDHHMMSS in Coordinated Universal Time (UTC). When this argument is omitted, the default end_time is 30 days from the start_time by default (see –s).
- -E *engine*: command option to use crypto hardware (OpenSSL engine) for zone signing using keys from a secure keystore when supported. The default *engine* is pkcs11 when compiled with PKCS #11 support or none otherwise.
- -f *file*: specifies the name of the file of the signed zone. If omitted, the default is the current zone file name appended with signed.
- -g: indicates that delegation signer resource records, which authenticate signed child zones, should be created; the resulting ds-set keyset can be provided to the parent zone's administrator for inclusion in the parent zone for signature.
- -h: prints help summary of this command.
- -H *iterations*: when generating an NSEC3 chain (when specifying the -3 option), use *iterations* iterations.
- -i *interval*: when resigning a zone (passing a previously signed zone in as input), the interval specifies the time interval from the current time for which any signature records expiring before the interval will be regenerated. Thus, if signature (RRSIG) records are set of expire within five days and the zone is resigned with an interval of six days, the signature records will be regenerated; otherwise, the current signatures will be retained.
- -I *input-format*: defines the *input-format* of the zone file to sign, either text (the default) or raw. Setting this option to raw facilitates signing of raw zone data, which includes dynamic updates and thus adds little value for static zones.
- -j *jitter*: enables specification of a window used to randomize RRSIG signature expiration times to reduce the impact of several simultaneous expirations, each of which would require signature regeneration when the signed zone is passed for resigning.
- -k *key*: the *key* specified is a KSK; multiple –k arguments may be provided.
- -K *directory*: defines the directory in which the key files are located.
- -l *domain*: generate a DLV keyset file; this keyset can be registered with the DLV registry to validate "delegation" for this zone.

- -n*threads*: specifies the number of CPU threads to use when performing this operation.
- -N *serial-format*: specifies the format of the SOA record serial number of the signed zone to either
  - keep: do not modify the serial number of the zone file input.
  - increment: increment the serial number in accordance with RFC 1982 serial number arithmetic.
  - unixtime: set the serial number to the number of seconds since epoch (since midnight UTC January 1, 1970 not counting leap seconds).
- -o *origin*: specifies the zone origin for the zone being signed.
- -O *output-format*: specifies the *output-format* of the signed zone as either text (the default) or raw.
- -p: use pseudorandom data when signing the zone, which is faster but less secure than using real random data per the –r argument.
- -P: disables the default postsigning verification tests, which include verifying that a valid nonrevoked KSK exists for each algorithm in use, that all revoked KSKs are self-signed and that all records in the zone are signed for each algorithm.
- -r *randomsource*: indicates a source of random data such as a file or character device such as the keyboard.
- -s *start_time*: specifies the date and time when resource record set signature records (RRSIG) become valid. The start_time may be specified relative to the current time using $+N$, where N is the number of seconds from the current time, or in absolute time using the format YYYYMMDDHHMMSS in Coordinated Universal Time (UTC). When this argument is omitted, the default start_time is 1 h prior to the current time to allow for clock skew.
- -S: "smart signing" leveraging key meta data, configured using the timing options of dnssec-keygen; searches the key repository for keys matching the zone being signed, includes them within the zone file in accordance with respective meta data and timing then signs the zone. Keys where the current date is past the activation or revocation dates but prior to retirement or deletion (or if no meta data exists), are used to sign the zone; keys where the current date is past the publish date but prior to other dates are published but not used to sign the zone.
- -t: print statistics upon completion of the signature process.
- -T *ttl*: defines the TTL value to use with DNSKEY records imported into the zone file from the key repository (if the TTL is not specified on any extant DNSKEY records in the zone), as part of smart signing (see –S).
- -u: update the NSEC[3] chain within the zone; also enables switching from a NSEC chained zone to an NSEC3 chained zone and vice versa depending on the presence of the NSEC3PARAM record in the zone file.
- -v *level*: sets the debug level.
- -x: sign the zone's DNSKEY RRSet with KSKs not additionally ZSKs.

- `-z`: ignore the KSK flag (SEP flag bit) when determining what to sign; that is, use the KSK [and ZSK] to sign zone RRSets.
- `zone_file`: the name of the zone file to sign.
- `key`: the keys to use to sign the zone data.

The output of the dnssec-signzone utility is the signed zone which uses the same name as the original unsigned zone, concatenated with a ".signed" suffix. Following our example, you can see below that the db.ipamworldwide.com.signed file is much larger than our original zone file. Consider our initial db.ipamworldwide.com file prior to signing

```
$TTL86400
ipamworldwide.com. 1D IN  SOA extdns1.ipamworldwide.com.
      dnsadmin.ipamworldwide.com. (
                    204 ; serial
                    3H  ; refresh
                    15  ; retry
                    1w  ; expire
                    3h  ; minimum
                   )
ipamworldwide.com.  86400 IN NS  extdns1.ipamworldwide.com.
                    86400 IN NS  extdns2.ipamworldwide.com.
                    86400 IN NS  extdns3.ipamworldwide.com.

extdns1.ipamworldwide.com. 86400  IN  A  192.0.2.34
                    86400  IN  AAAA 2001:db8:4af0:2010::a
extdns2.ipamworldwide.com.86400  IN  A  192.0.2.42
                    86400  IN  AAAA 2001:db8:4af0:2011::11
extdns3.ipamworldwide.com.86400  IN  A  192.0.2.50
                    86400  IN  AAAA 2001:db8:4af0:2006::9

eng.ipamworldwide.com. 1w  IN  NS  ns1.eng.ipamworldwide.com.
                       1w  IN  NS  ns2.eng.ipamworldwide.com.
ns1.eng.ipamworldwide.com.  1w  IN  AAAA 2001:db8:4af0:2007::7
ns1.eng.ipamworldwide.com.  1w  IN  AAAA 2001:db8:4af0:2009::12

$ORIGIN ipamworldwide.com.
             1D  IN MX 10 smtp1.ipamworldwide.com.
             1D  IN MX 20 smtp2.ipamworldwide.com.
```

```
www           1D  IN  A 192.0.2.37
              1D  IN AAAA 2001:db8:4af0:2010::25
              1D  IN  A 192.0.2.53
              1D  IN AAAA 2001:db8:4af0:2006::5
w3            1D  IN  CNAME www.ipamworldwide.com.

smtp1         1D  IN  A 192.0.2.36
              1D  IN AAAA 2001:db8:4af0:2010::1b
smtp2        1D  IN  A 192.0.2.45
             1D  IN AAAA 2001:db8:4af0:2011::2b
ftp-support 1D  IN  A 192.0.2.44
             1D  IN AAAA 2001:db8:4af0:2011::2c

$INCLUDE Kipamworldwide.com.+005+14522.key
$INCLUDE Kipamworldwide.com.+005+06082.key
```

Contrast this with the signed version:

```
ipamworldwide.com. 86400   IN SOA extdns1.ipamworldwide.com.
dnsadmin.ipamworldwide.com. (
                           204       ; serial
                           10800     ; refresh (3 hours)
                           15        ; retry (15 seconds)
                           604800    ; expire (1 week)
                           10800     ; minimum (3 hours)
                           )
             86400   RRSIG  SOA 5 2 86400 20100305135354 (
                           20100203135354 14522 ipamworldwide.com.
                           OQS+AaE57+ffRfz+SaMHOJI6b4l2bNnsSDIK
                           mIIMdmXOw8cylCMieaUBz8ek64FyMWLGh2c5
                           HogVxtt7s9cHICosxqhqZNXYT7GP+YpRRVO4
                           uCGgq6uoqCpgj1L39tqnSQ1da8pT5a6DRCIJ
                           fqsS5ubrmA/20cc02c15XFTlAik= )
             86400   NS     extdns1.ipamworldwide.com.
             86400   NS     extdns2.ipamworldwide.com.
             86400   NS     extdns3.ipamworldwide.com.
             86400   RRSIG  NS 5 2 86400 20100305135354 (
```

```
                20100203135354 14522 ipamworldwide.com.
                qVdOx6s9IAL4YWz2hPB1Q5aVNPcPbIsREenD
                PP/7GyXbQKxAdDDugaWPHoKEvPA9f1SBWomZ
                h4pGOKJaA5Pk9okF3FkHLHclTFVGfhTEdrVj
                Dk6a8eRNoU+CMHWwmfJtNFpYpVVd6Ch1LWdw
                ZJ27Z80HZrHtwZ8XmubPzu8MZlE= )
86400   MX      10 smtp1.ipamworldwide.com.
86400   MX      20 smtp2.ipamworldwide.com.
86400   RRSIG   MX 5 2 86400 20100305135354 (
                20100203135354 14522 ipamworldwide.com.
                dR4kJtp5DyvCHTF7+uCNloKCRNVx5jM/XOd9
                H5F7OhnDUIgPWKYnuCbL3PBhx1iK9OnrrL1g
                ZvEuTAvifzzax4n8CSPCB0CbrMWWUXQ44vKG
                I0W0LwzQKJXlPGHzGiG+6dktfqOnBgppXekA
                QWBJA6nOAeGKtqQMtKUa75uqs2Y= )
10800   NSEC    eng.ipamworldwide.com. NS SOA MX RRSIG
                  NSEC DNSKEY
10800   RRSIG   NSEC 5 2 10800 20100305135354 (
                20100203135354 14522 ipamworldwide.com.
                WyZl4AduBUWdED01Ckc+I0nSArek5n3r6rKX
                m26H5Sjow/RSpgmPJfGOH/9gjyEwnGoqrKbh
                5s7kxtnvF3xVYFE1If7zv5bHxSvBqMDqdNXq
                ChY9BJ9kOemQ0L7NlpreadXfyVXBthl5jaPC
                vKLSwAjmNAzbtV4f6S+CIDK288w= )
86400   DNSKEY  256 3 5 (
                BQEAAAABx6QAwJiz6hHa/eUI2pGz6rvwEYpJ
                di1TJH8Uj4lDPTmzseCOgFEqB3/dZB0Q5LEs
                1ZetAJJEk4F+WccRKwqnIcGkvIKfTC8hn+gb
                iBAnadQRFLxNMBs6KB0e+yqiNK60sbrn22F8
                AYRiG3n2rTQndVtkaZep9jbcCqfu/DagB10=
                ) ; key id = 14522
86400   DNSKEY  257 3 5 (
                AwEAAdSAwGoUBhtjpE8GLGN4ryt8yEq71Dqd
                E+ij3boe9lmvpM02YZ1/AQxoHbyA7NqRr+8d
                sTM8OrF2yFRbcP1y0/9q37T0PqxL5HjAZ8Hr
                DoW9R/pC3XyRe9pMzRNr4as+c/xEISfhxzvR
```

```
                                84CndF5XvFeh3H0kVDeTb+7QRrG7hnph4P8w
                                4SMg76tBvxHLFmj3OdP8vIUpRAnexEAdclam
                                j1ZSPjLcdICzpDvQB/LLsYxx8wx2h0vTvhxZ
                                klqmy1dPBtIZu2A551VIrU0xgCJxDjJGCgBb
                                rp1C01tYSdqlA1I2HCL8eV7io/CxnCuSThPl
                                XaPLySojJpXUgDomWgVYeo0=
                                ) ; key id = 6082
                86400   RRSIG   DNSKEY 5 2 86400 20100305135354 (
                                20100203135354 14522 ipamworldwide.com.
                                V0bEwZmY56OrGQb02B/Pf17RAcFyPZAvPT/W
                                Rm/+nluSOYMVqdzRaKM/ae47KslioXm3tNcy
                                GF3uBvBql7xPzIOuIy3COoorXmbsshbuANo7
                                YfQsyXWuX2BIjjLAVRRLQolVcdDyyleoA0E7
                                BebPM+fQQtvN2C2IjrcacJyeUlc= )
                86400   RRSIG   DNSKEY 5 2 86400 20100305135354 (
                                20100203135354 6082 ipamworldwide.com.
                                e8jCEVY6C1lSImGqjgzVWAgp7cC4AWuntFvc
                                oCCO+2BwGxe7+zxP2r02CCSOCIrTqtgwpNRd
                                5aH4xBrYmZh0IFQ7OxTFSGBvQ4DxC8ZDdQVS
                                uTYCBSzN7kXRJZopZv3chhf7/9uyz3gqtQnl
                                5RyUVATMOG5eu+ewBFqGIsXJv5XMNG7ZTO15
                                rtRd8zF/7MIY7TlSbHULGP7OJxcNFtyt8wnc
                                /dObfcxril4tOwLPVF4QnLnLxAHvdWt+QPVQ
                                z23WIc0U+rg6U6FsSjoi0U2QAxVFebenTJED
                                U2juAdqEE8I1Y9oOvNQVtYFFjXFgi1vDLGCG
                                zM8i4fI9uGZUHvzKng== )
ns1.eng.ipamworldwide.com. 604800 IN AAAA 2001:db8:4af0:2007::7
ns2.eng.ipamworldwide.com. 604800 IN AAAA 2001:db8:4af0:2009::12
eng.ipamworldwide.com. 604800 IN NS  ns1.eng.ipamworldwide.com.
                604800  IN NS   ns2.eng.ipamworldwide.com.
                10800   NSEC    extdns1.ipamworldwide.com. NS RRSIG NSEC
                10800   RRSIG   NSEC 5 3 10800 20100305135354 (
                                20100203135354 14522 ipamworldwide.com.
                                dWwY0rZRfW5aYgBsbRuCxot6CGGG8hfgHId7
                                84IZIYi9HHgr02saBdlzmzqJCGre0pGSDBvf
```

```
                        ZpJP1BVUS1NuMycEBFBUIS8IUASDTxcLGjrT
                        169vIqiyXjICzrsu2fzKL1QNwUOFMGiedglh
                        1jkUJ1jKKs9yr4XFZBwP/y8OpoQ= )
extdns1.ipamworldwide.com. 86400 IN A192.0.2.34
            86400   RRSIG  A 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        IwNfRz7m6Rneh6hpacdIpTHGRftsU8e931OP
                        bjC0Dfw92DXn51uHghiCoE+rrO4zK1wYFP5L
                        CoKF43whVX1EXOt7UFGuAebr4587DnDqhKol
                        9XivKc35HvPz1ErniZHuUIsZCjvuziwvGIXS
                        72PkoHzNw/lxv+nDriemFn7tWxE= )
              86400 AAAA   2001:db8:4af0:2010::a
              86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        aNzJgdLi4DTttIUj+Y+9FLI2eAu5iRX9yewN
                        jvFG3aJ4moO4fWwhKFynltcfJFpKjHyq4eCD
                        PamIS/9fDOn8OdX1g8CkfKNQIszUoAkhSQXH
                        6avko1jwgP0lqHwjRNhdcW2UuE+pjyvgNlTW
                        ZOgb65nR+UjSJQXRQnHpyhyD+nk= )
              10800 NSEC   extdns2.ipamworldwide.com. A AAAA RRSIG
                             NSEC
              10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        ipB8eo8GLPvbCCzUF6ETXBiXsRXZiWu8y21z
                        uEoxJn+3T9dYXFEFFpdyj5Qnhl/gnvwpc1mP
                        sFyg0+P5mNziXO/Aj3LQF2HJMnQxT34dQdJb
                        Ze/6KBJZO6KZXwXrQXxVGrbFHY9xY5Q0gfs4
                        J2MUAZB074KWOVZKUzLUczgrwhI= )
extdns2.ipamworldwide.com. 86400 IN A192.0.2.42
              86400 RRSIG  A 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        ax6Umlog3DSn+KxIQSvbQjES9CwuaYZ+G0yT
                        NHOIwVOrV4cjP7LA2Pc2p7bQjwoTMkXK5uoU
                        Or8Mnd7/boJyQUrBF62pbhOqJ9mKbvrYD1ud
                        SivEiDnxAv0FTwagCe22Vvd3DNTjXUhizBt7
                        DlIbA92lSCiNHqeFT/OljqcW+Z0= )
```

```
                86400 AAAA   2001:db8:4af0:2011::11
                86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                             20100203135354 14522 ipamworldwide.com.
                             aQO0ipvwjtAS0DZiXJoTot9iPAToI5rqrkMD
                             lXRNimxuT/ED0+S94OUg5rA5a/XS80aDFSyD
                             uqLIVIiZC4Zd5jHazPxEjJR7YyJ0sx8kIy5Q
                             85LBJQhVsiADcoKz7NZ8TRFzSEGQNKMLVYIx
                             kVx8JpJcGWLeXBekk5J46OeacfE= )
                10800 NSEC   extdns3.ipamworldwide.com. A AAAA RRSIG
                               NSEC
                10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                             20100203135354 14522 ipamworldwide.com.
                             J8j82DSNwUc0M2dd2vPzkTlOnjxrrTeKIWH2
                             h13hjbH3xr18WLQdJQiqJpXapXSKGX/57+C8
                             EO+OBbsqNMpwf+bNhxdnJazB7elYdk7KI8Xp
                             TmpyV9zRTJjr3U3l6pw2GjaCMkBDw8JD1+6w
                             LJjib4JgHg3pDswvo6ShXxpnezk= )
extdns3.ipamworldwide.com. 86400 IN A192.0.2.50
                86400 RRSIG  A 5 3 86400 20100305135354 (
                             20100203135354 14522 ipamworldwide.com.
                             a6IVQOXfc0UgsIfCJA/yGDvPdXUrXH2HJzS9
                             h/DGEIdu3ZBNcEwtKVvd4ph/rHXknX2Ito2m
                             4/1OLtvFdriZjhbpIERCatl45ySxhvugbZlb
                             EAjEWa1kixmPoOtXZ+pAS+7cLCxkodr5Np2t
                             f9Ppdv5bx4/a9BfM8abrUwrT988= )
                86400 AAAA   2001:db8:4af0:2006::9
                86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                             20100203135354 14522 ipamworldwide.com.
                             AMeurMSeauKG/w0KSgo9tKWToMDXEOtArCmu
                             l3VKDUDN22Y7yfIUX+nwcUJuLRU4tLfeiLBT
                             E8IIjsJ3Qu9SQmCBB/4VCHjNax98c4+/RBym
                             M9sKuprQK9MEzV5kqqYyHdVuPFzSWCp0QXCO
                             AWrWGWfkO3oXS6oj+gqK3hHnAsQ= )
                10800 NSEC   ftp-support.ipamworldwide.com. A AAAA
                               RRSIG NSEC
                10800 RRSIG  NSEC 5 3 10800 20100305135354 (
```

```
                        20100203135354 14522 ipamworldwide.com.
                        nfQMcp6s2IyVItiCmb89DiSKYmdurlBo0Nx3
                        0IQYcoMvZVVXMa4ynCoq3lKdebjhGrW8e6NG
                        c5SyPYrBzjw1NVEPIr1mNoVN2EEBqquPYluC
                        z9f0M5N534yThP01yCsjee7FpIXGKYObhb5+
                        i5wLH10NrIpLJEAw3oWsXNPxkhQ= )
ftp-support.ipamworldwide.com. 86400 INA 192.0.2.44
             86400 RRSIG  A 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        umyIYTUI2YaFXcRp9xATrAK7YnOz/PCbzOSF
                        xJLL9CLNzmtdPEvFW7iO9oeC8C+R3WfYafhV
                        aWiT/BYPbwqvxHaxRWFJ7hIO87n5PHfAHxyE
                        dIr1lLZO5f0IKK8oIgawIyHbE/XeqYHVeZpY
                        zJSGGMBiyTI/VGKluud17+/EDh4= )
             86400 AAAA   2001:db8:4af0:2011::2c
             86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        A43jiBaDMIhL9KtqP2uE2iG+sn5SZBVhqK9Q
                        ChRR0512pZJ5WGPip0KjgcJxaVnMbbBuyM7v
                        lzW6G1PerBwtbaX/zi2YnW+O0XyBYGlXjXPC
                        bHjM3I7Z07WgHD/I4jrHZVQczUDSmZCJQBIK
                        zEYlTt+su4K6EIfxw3uBlrheAAc= )
             10800 NSEC   smtp1.ipamworldwide.com. A AAAA RRSIG
                          NSEC
             10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        v/LRbW7drv03r+F5XasqZ2bjdGXQ7VP6kvOa
                        gt3s/gT5W/c8aLfTeA3lmwwEk3DrNEB9U+MV
                        XE9YdI1iLySu8J07hF9qJfSiCSIkZgmf5UDZ
                        BUUKifIXZVRHUy8uD2pXP3btZOrhR9CXU5oE
                        EfrvaGv7++yC+IhRJN7pbg+WEU0= )
smtp1.ipamworldwide.com. 86400   IN A    192.0.2.36
             86400 RRSIG  A 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        lSISPwoCpLdSfWFFjhfuASY72DoA06dMPAic
                        5vhRJWQfoUbisWrGt29z7r7S7XYIwgRARURO
```

```
                        JDUSe93z7TzbjxO4UPDbuheFDYI7r+vDXLj2
                        cQgKT4gPJ6UCi2kawWaVbAzPz+ZzV2gfxJfc
                        fsjARB5rbNDk1BOO6IDI3pfPYh0= )
                86400 AAAA   2001:db8:4af0:2010::1b
                86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        Jap9zaU4gWcxHzXmtkK8NtCKGUCE/AdPf+/d
                        yWJC5PG7ClildQsxCIhbvgLHdQ0YfFMN5nvd
                        abt3fybBoTtbNATZeBqFDalMnF3IBzyhChA+
                        0DC1R27LGk7iyOZ5zsq055ZgROpkBbML3o9k
                        M7Y+Lx+nM3j44zj6YoUDsAUvP1s= )
                10800 NSEC   smtp2.ipamworldwide.com. A AAAA RRSIG
                          NSEC
                10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        x1YlFJQBUhSOTB/T7nrntcaB7x96AK+AAJZT
                        787XIryUwg5boDkA5MOGNxAoL6nurtbi3+6f
                        GLDoG4HYLsEmJlamw9+IANm1u2yLsg5q2viL
                        1ymroI0AlpeXNptDevgZ5+CiRiRKkNw0+BZ1
                        YCrdNJTBUo8pYfZDxdBpihi87EU= )
smtp2.ipamworldwide.com. 86400   IN A   192.0.2.45
                86400 RRSIG  A 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        a+qfAnPTIcI7nBNRhg6BDZrFuQvBbiLZUPXA
                        kSXeLNkwtK5bodr+j0nZQqUFsCvHw/Gj2FH7
                        7L2ROcDto0QHE9WwKy3AjNtvRGg/GK54uO2v
                        A4NEx8C0sgIyWPkIC9Nbndp4bE2zV1r3O4Wr
                        UkAGYtD/ZMv79vhB8AsLKyfS+yM= )
                86400 AAAA   2001:db8:4af0:2011::2b
                86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                        20100203135354 14522 ipamworldwide.com.
                        jQrvFnE/4JqFSfl6b/GR8j2hv/B+4XmuslCM
                        4P2D6YRYGNhZCeOw4DY3U9fGsg+B8gZii7U+
                        Rc9Qe8RyzV+wu8gy+65uvbSl9sb6zfGOrOp2
                        P+ZsAy7R0ZtPjzEdMLAIJdea4LdAgUO9IqNo
                        Q5ro79H9GAHptAw2epa+1XAp+wc= )
```

```
            10800 NSEC   w3.ipamworldwide.com. A AAAA RRSIG NSEC
            10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                         20100203135354 14522 ipamworldwide.com.
                         gKXAbEocdwlnPIo9YtwLwOatBlamwpQTEM+e
                         rKjgibrjYP1ymBiRwOs8lnrfXxCbv6v3ix6Q
                         IQcQrzKPugVEaIxUl6kqqH0LsXYwgbixppQX
                         Lcn1z9Wfmdv23Z6njQrdR+DmF6aQZPlUaiTn
                         SPtE26w59U1rtcyhm2p0vqic5l8= )
w3.ipamworldwide.com.  86400  IN CNAME www.ipamworldwide.com.
            86400 RRSIG  CNAME 5 3 86400 20100305135354 (
                         20100203135354 14522 ipamworldwide.com.
                         PTRpKmkZhBw18c1ZscxIlCL7P23fYgZBsEX9
                         DKrawjkyMZ5B+EqQaGNdsfgvmvirrcxCzv/K
                         MaMaeWBGI4Bb9gykm72thXneud5gHi5zLjsK
                         4uK/r3EdWcurQ89R44Q7pTUOy4yzCdpH/KDE
                         Qjd6P/JrWLnO/WW0gxurwHPFiBE= )
            10800 NSEC   www.ipamworldwide.com. CNAME RRSIG NSEC
            10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                         20100203135354 14522 ipamworldwide.com.
                         W0jcsvdSJfLuJ617fAUdfnat9fd45OE6toO7
                         GHwkjRPWm1Q9C83W0Bgc161gZ4r/q1ZKoE6b
                         FvFOIIxz4NhPiJVb2bUbSL6A1K0vwD4KUGL4
                         ExKfuPMl6/gLOE/Tqczcp/2ETXm5yksOkoPJ
                         ynOMoLpBIyNlHeJwtPBaHlbE+B0= )
www.ipamworldwide.com. 86400  IN A   192.0.2.37
            86400 IN A   192.0.2.53
            86400 RRSIG  A 5 3 86400 20100305135354 (
                         20100203135354 14522 ipamworldwide.com.
                         xjwhHWIy25aOvLP2E1y9aaN6GRcGUxoN4o+P
                         eZ0Wc05zjlDu6o1ZOCXivrbIOP4LVS7pqMX3
                         bg4SQDmzmRDQ0H/+Q8Fzxbf1UFQNcVB2uhtV
                         6R8DfNwRwIugoL+33qE2MOrrxWz16JutI2qo
                         vkYogNqDj1MNiiKkoGgmJQmiHYc= )
            86400 AAAA   2001:db8:4af0:2010::25
            86400 AAAA   2001:db8:4af0:2006::5
            86400 RRSIG  AAAA 5 3 86400 20100305135354 (
                         20100203135354 14522 ipamworldwide.com.
```

```
                    lRoCDp+0y/HM/xyEdciqO5cDWcRzxmQCwPbs
                    GKrCe+OoYHfTFnSBCAEReY4tneb/HMwYbqxV
                    SRp5oW2FPDi5GZunL7tLp7gF0tF7M9XlJVmi
                    9PDg9wiNzDxw/CgbsN/wbtsRpgbPxQwkACiP
                    eRsNDL3Y5EAxLi24yFw+Qay6uEc= )
10800 NSEC   ipamworldwide.com. A AAAA RRSIG NSEC
10800 RRSIG  NSEC 5 3 10800 20100305135354 (
                    20100203135354 14522 ipamworldwide.com.
                    auNzMg6x34+oradbjFKoQquKmB8sAmKg44FF
                    8FCuh7FI/mrKNHVuv1YmVNXNK/ZHA1JpVYzH
                    fpe4KxPGh8IcDftEfqd52Z0LsetYeRvxNzxQ
                    sAS+OzClCIiTiEpUNte6siExj7YvhBlPN4e4
                    pnkzTKPULWat489Juzo2U77XysA= )
```

Needless to say, signing a zone increases its size tremendously! It also increases resolution packet sizes, given the extra RRSIG and NSEC information with each RRSet, not to mention the potential for additional message traffic to validate the chain of trust back to a trust anchor.

Referring back to our discussion of the digital signature process, the original resolution data is of course the "data" from Figure 13.1 to be signed. The data actually consists of the entire RRSet which is hashed then signed using the private key of the key pair. The resulting signature comprises the signature field of each RRSIG record. Thus, at the beginning of our signed file, we have our original SOA record, followed by its corresponding signature (RRSIG). Then our three NS records are listed. This RRSet comprised of these three records is signed per the following RRSIG record. Likewise,the MX RRSet is listed and signed. Notice the RRSIG records indicate signature using the ZSK, per the key tag field value of 14522. The DNSKEY RRSet is itself signed by both the KSK and ZSK as evidenced by the two RRSIG records with respective KSK and ZSK key tags. Usually the KSK only signs the DNSKEY RRSet with the ZSK signs all zone RRSets. Notice also that the ns1 and ns2.eng.ipamworldwide.com glue records are not signed as these records are authoritative in the eng.ipamworldwide.com zone, not the ipamworldwide.com zone.

The NSEC record listed next provides a canonical ordering of records to identify and authenticate a negative answer for a non-existent resource record. This particular record indicates that the next owner is eng.ipamworldwide.com. This NSEC record also is signed. Each of the remaining RRSets includes an NSEC record and RRSet signature (RRSIG record).

13.3.4 Link the Chain of Trust

Now that the zone has been signed, you should determine its place in the chain of trust. That is, determine if the parent zone is signed or not. If the parent zone is not signed and

the newly signed zone is the top-level domain, that is, signed (i.e., *zone apex*; e.g., `com` is not signed as of this writing.), recursive resolvers querying on behalf of stub resolvers must be configured with the zone's public KSK as a *trusted key*. This informs the resolver that zone information signed with this key is to be trusted. For those resolvers, which trust our ipamworldwide.com zone administrators' data, the KSK 06082 public key can be configured within the respective trusted-keys statement in each recursive server's named.conf file as per the following

```
trusted-keys {
"ipamworldwide.com." 257 3 5
  "AwEAAdSAwGoUBhtjpE8GLGN4ryt8yEq71DqdE+ij3boe9lmvpM02YZ1/
  AQxoHbyA7NqRr+8dsTM8OrF2yFRbcP1y0/9q37T0PqxL5HjAZ8HrDoW9
  R/pC3XyRe9pMzRNr4as+c/xEISfhxzvR84CndF5XvFeh3H0kVDeTb+7Q
  RrG7hnph4P8w4SMg76tBvxHLFmj3OdP8vIUpRAnexEAdclamj1ZSPjLc
  dICzpDvQB/LLsYxx8wx2h0vTvhxZklqmy1dPBtIZu2A551VIrU0xgCJx
  DjJGCgBbrp1C01tYSdqlA1I2HCL8eV7io/CxnCuSThPlXaPLySojJpXU
   gDomWgVYeo0=";
};
```

Within the recursive server configuration, we have declared a trust anchor or SEP at the ipamworldwide.com zone. Note that a trusted key entry with the corresponding KSK public key is required for each trust anchor you wish to configure. As we'll discuss later, the more trust anchors you configure, the more keys you need to manage during each zone's key rollover process. With the signed root and TLD zones, only one trust anchor need be maintained.

Automated trust anchor update capabilities reduce the manual management of trust anchor rollovers. For such trust anchors, instead of using the `trusted-keys` statement, the `managed-keys` statement can be used. In the following example, we use the public KSK from our trust anchor as the initial key. This key serves as the trusted key initially, but as the zone administrator for the ipamworldwide.com zone publishes, activates, revokes, retires, and deletes keys in accordance with the timing capabilities and automation of BIND 9.7 and above, this recursive server will remain in step and keep its own repository of current trust anchor keys per trust anchor. Hence, when this initial key is revoked and another key activated, signature of the DNSKEY RRSet by both the newly activated and this now-revoked key validates the transition to the newly active key.

```
managed-keys {
"ipamworldwide.com." initial-key 257 3 5
    "AwEAAdSAwGoUBhtjpE8GLGN4ryt8yEq71DqdE+ij3boe9lmvpM02YZ1/
     AQxoHbyA7NqRr+8dsTM8OrF2yFRbcP1y0/9q37T0PqxL5HjAZ8HrDoW9
     R/pC3XyRe9pMzRNr4as+c/xEISfhxzvR84CndF5XvFeh3H0kVDeTb+7Q
```

```
    RrG7hnph4P8w4SMg76tBvxHLFmj3OdP8vIUpRAnexEAdclamj1ZSPjLc
    dICzpDvQB/LLsYxx8wx2h0vTvhxZklqmy1dPBtIZu2A551VIrU0xgCJx
    DjJGCgBbrp1C01tYSdqlA1I2HCL8eV7io/CxnCuSThPlXaPLySojJpXU
    gDomWgVYeo0='';
};
```

Now if the zone just signed is a child zone of a signed parent zone, the parent zone administrator must include the delegation signer record in the parent zone file to link the chain of trust. In this manner, the parent zone can vouch for this signed child zone. Thus, trust anchors need not be configured in resolvers or recursive servers for this zone, just its parent or an even higher level ancestor signed zone.

The –g option on the dnssec-signzone utility automatically created our DS records for the zone in a dsset-ipamworldwide.com. file. The file contains two DS records, one of which may be chosen based on the preferred digest type. The integer shown before the digest in the examples below indicates the digest type. Type 1 is SHA-1 and type 2 is SHA-256. The digest follows which is computed as a hash using the corresponding digest type or algorithm of the signed zone's KSK DNSKEY resource record owner and RData fields (i.e., the KSK DNSKEY record, omitting the TTL, class and type).

```
ipamworldwide.com.
INDS6082515F696637B085D8F5CBFD0C8B9E031CB6CB07159B
ipamworldwide.com. IN DS 6082 5 2
7FFD9203E916B5D49F631D060FAFD05D26974BEFCED25AACB88122722E4A7AA9
```

In terms of authenticating records or their nonexistence in signed child (delegated) zones, the delegation signer resource record type provides the link from a parent to a delegated child zone's key as a link within the chain of trust. We'll walk through how this works within the resolution process next. The DS resource record has the following format

Owner	TTL	Class	Type	RData			
Delegated domain	TTL	IN	DS	Key tag	Alg.	Type	Digest
ipamworldwide.com.	86400	IN	DS	6082	5	1	5F695D8F5BFD0C...

The RData portion of the DS record identifies the key tag or id of the child zone's public KSK, while the Algorithm matches the Algorithm field in the referenced DNSKEY record. The Digest Type indicates the type of hash or digest which is conveyed in the Digest field. Valid Digest Type values are 1 (SHA-1) or 2 (SHA-256). The Digest field contains the digest or hash of the corresponding child zone public KSK DNSKEY resource record owner field concatenated with the same DNSKEY record's RData field.

The parent zone administrator would add the DS RRSet to the parent zone and sign it to authenticate its origin and integrity.

In BIND 9.6, a new utility, `dnssec-dsfromkey`, was introduced. This utility enables generation of DS resource records without having to re-sign the zone using `dnssec-signzone`. This utility is available with the following parameters

- `-1`: use SHA-1 as the digest algorithm
- `-2`: use SHA-256 as the digest algorithm
- `-a` *`algorithm`*: where algorithm may be
  - SHA-1
  - SHA-256
- `-A`: include ZSKs along with KSKs for generation of DS records; if omitted, only DS records for KSKs are created.
- `-c` *`class`*: identifies the class (default is IN).
- `-d` *`directory`*: directory location of the keyset files.
- `-f` *`file`*: specifies a zone file name in lieu of specifying the keyfile name.
- `-l` *`domain`*: generate a DLV set instead of a DS set and append *domain* to each record in the set.
- `-K` *`directory`*: defines the directory in which the key files are located.
- `-s`: command argument is a domain name not a keyfile name.
- `-v` *`level`*: specifies the debug level.

The dnssec-dsfromkey utility can generate DS or DLV records based on a keyfile or a domain name; the –s argument defines the argument as a domain name

```
dnssec-dsfromkey -s [-v level] [-1] [-2] [-a algorithm] [-l domain]
keyfile
```

The omission of –s identifies the argument as a keyfile name.

```
dnssec-dsfromkey [-v level] [-1] [-2] [-a algorithm] [-l domain] [-K
directory] [-c class] [-f file] [-A] domainname
```

13.4 THE DNSSEC RESOLUTION PROCESS

Now let's review how the resolution and verification process works. Configuring the trusted or managed keys statement above into our recursive server configuration (named.conf), we have declared ipamworldwide.com as a trusted zone. When I issue a query for a host within the ipamworldwide.com zone, for example, `ftp-support.ipamworldwide.com`, my resolver will set the DNSSEC OK (DO) bit in the EDNS0 extended Rcode field. DNSSEC requires EDNS0 to support this extended Rcode field

and also for the generally large response packets likely exceeding the nominal 512-byte UDP packet limit. The packet length increase is due to the response by the server configured with the authoritative signed zone with not only the resolution data requested, the A record(s) for `ftp-support.ipamworldwide.com`, but the associated signature record associated with the A record set.

13.4.1 Verify the Signature

The signature process signs resource record sets, which are groupings of resource records with common owner name, class, and type. The signature is created using the private key referenced by the key tag parameter and is placed within the signature field of the RRSIG resource record.

The RRSIG resource record has the following format

Owner	TTL	Class	Type	RData								
RRSet Owner	TTL	IN	RRSIG	Type Cov.	Alg.	Labels	Orig. TTL	Expire	Inception	Key tag	Signer	Signature
ftp-support.ipamworldwide.com.	86400	IN	RRSIG	A	5	3	86400	20100305215354	20100203215354	14522	ipamworldwide.com.	umyI...

The RData fields within the RRSIG record are defined as follows.

- *Type Covered.* The type of the resource record set covered by this signature. In our example, the A record type is covered by this signature which signs our two-resource record RRSet with owner ftp-support.ipamworldwide.com.
- *Algorithm.* The algorithm used in generating the key, which is encoded in the same manner as the Algorithm field of the DNSKEY resource record type (see DNSKEY above).
- *Number of Labels.* Indicates the number labels within the owner field. For example, ftp-support.ipamworldwide.com has three labels. This field is used to reconstruct the original owner name used to create the signature in the case where the owner name returned by the server has a wildcard label (*).
- *Original TTL.* The TTL of the signed RRSet as defined in the authoritative zone, used to validate a signature. This field is needed because the TTL field returned in the original response is normally decremented by a caching resolver and use of the TTL field may lead to erroneous calculations.
- *Signature Expiration.* The date and time of the expiration of this signature expressed as either the number of seconds since January 1, 1970 00:00:00 UTC or in the form of YYYYMMDDHHmmSS where

  - YYYY is the year (within 68 years of the present date to prevent numerical wrapping of this field)
  - MM is the month, 01–12
  - DD is the day of the month, 01–31
  - HH is the hour in 24 h notation, 00–23
  - mm is the minute, 00–59
  - SS is the second, 00–59
  - Signatures are not valid after this date/time.
- *Signature Inception*. The date and time of the inception of this signature formatted in the same manner as the Signature Expiration field. Signatures are not valid before this date/time.
- *Key Tag*. Provides an association with the corresponding key (DNSKEY resource record(s)) by key id or tag.
- *Signer's Name*. Identifies the owner name of the DNSKEY resource record that was used to create this signature.
- *Signature*. The cryptographic signature covering the RRSIG RData (excluding this Signature field itself) concatenated with the resource records comprising the RRSet identified by the RRSIG owner, class, and covered type fields.

Thus, the response to our query includes the A records and the associated RRSIG record as indicated by this response captured with the `dig` utility below. The server will set the Authentic Data (AD) bit in the DNS header in the response only if it has authenticated (cryptographically verified) all included resource records in the Answer section and all included negative response resource records in the Authority section. Note that if you query the server which is authoritative for the issued query, the AD bit will not be set. This server simply returns the answer and leaves validation rightly to the querier. If you query your recursive server which is not authoritative for the queried information, it will perform the resolution and DNSSEC validation, which if successful, will set the AD bit in the result. We'll review the details of the dig utility, which is very useful in verifying and troubleshooting zone configurations in Chapter 14.

```
$ dig +dnssec A ftp-support.ipamworldwide.com. @127.0.0.1

; ≪ ≫ DiG 9.6.2 ≪ ≫ +dnssec A ftp-sf.ipamworldwide.com. @127.0.0.1
; (1 server found)
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 462
```

```
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;ftp-sf.ipamworldwide.com.    IN    A

;; ANSWER SECTION:
ftp-sf.ipamworldwide.com. 86400 IN    A    10.1.32.9
ftp-sf.ipamworldwide.com. 86400 IN    A    10.1.32.5
ftp-sf.ipamworldwide.com. 86400 IN    RRSIG  A 5 3 86400 20100525173519
  20100425173519 14522 ipamworldwide.com.owHoS6b1xTNKuzJjgJs3nL4Kwr-
  LehnfixVjAF2T6 RHu4dVmq4w1p+FNC Oji2BkWKOhjY3+7jU4doFr/RNioe8vmsqyn
  R5YeSSRzzFy/d63Riz3bQ5BANbGRqpTn6Q9HQlm+KYSpwY5CrjqOQnP+Ynme4nhT9
  +z8h5ahdwtK9 EtI=

;; AUTHORITY SECTION:
ipamworldwide.com.    86400  IN    NS    ns.ipamworldwide.com.
ipamworldwide.com.      86400    IN RRSIG   NS 5 2 86400 20100525173519
20100425173519  14522 ipamworldwide.com. OLonIvBmJZDEZoRRvOiq7GnlWnr-
  8LTWHtKSR60CJNl3hd23Vvkbq/EkV 46wp6OK6Q0qNtJGE+YqFW9xml7d6kQRZO-
  qIyCiDZqHQinV7LlAa0Da8z5+UGduD3gVLceES7lvGZpLlbyUm9kFGf5FhPZ/
  JciPF4qKUdAvfEeitu/aY=

;; ADDITIONAL SECTION:
ns.ipamworldwide.com.  86400  IN    A    10.1.32.4
ns.ipamworldwide.com.  86400  IN    RRSIG  A 5 3 86400 20100525173519
20100425173519 14522 ipamworldwide.com. IHtLJaWam57mVoYCgFqlEPC9N9p7n-
  Wicy7MBvdQP6PgNfhnOTOg2vQHR rQRDdBWBmgaSRoiWSdF2IQTEfH4T16591-
  OEjtBnPR/7zRAxU9abnkUDvGCZsAFfqKfWxBZFrxUTbxloekEhMC98FqCnvaRIsL-
  NYbiP/0KhehWmBF nlA=
```

The recursive server or resolver, having received the RRSet (data) and the RRSIG (signature) may then issue a DNSKEY query to obtain the DNSKEY RRSet if it is not already cached or provided with the response in the Additional section. The signature in the RRSIG record is processed with the key of tag 14522 and compared with a hash of the RRSIG RData (less the signature) concatenated with the resource records within the RRSet. If the comparison yields a match, the signature is validated successfully. Next, the RRSIG of the DNSKEY RRSet is used to validate the ZSK itself. Performing a similar calculation as just described to validate the A RRSet, the resolver or recursive server

validates the DNSKEY RRSet with respect to the public KSK signature. Given a successful match as well as the fact that the public KSK matches a configured trusted-key, we have therefore successfully validated the A RRSet data.

13.4.2 Authenticated Denial of Existence

What if I mistyped the hostname I intended to query? Without DNSSEC, I'd receive an error (NXDOMAIN) indicating the record was not found. Just as an affirmative answer may be spoofed, "not found" answers may be too. To address this potential vulnerability, DNSSEC incorporates the Next SECure (NSEC) resource record to provide a means to authenticate the nonexistence of a record matching the query. The NSEC record essentially points from one RRSet to the next within the zone file, identifying gaps between RRSets. The format of the NSEC record is as follows.

Owner	TTL	Class	Type	RData	
RRSet Owner	TTL	IN	NSEC	Next RRSet Owner	Type Bit Maps
ns1.ipamww.com.	86400	IN	NSEC	ns2.ipamww.com.	A AAAA RRSIG NSEC

In this example, the NSEC record associated with owner ns1.ipamww.com indicates that the next owner name in canonical order is ns2.ipamww.com, and this owner (ns1) exists with resource records of type A, AAAA, RRSIG, and NSEC. This record indicates that there aren't any records canonically between ns1.ipamworldwide.com and ns2.ipamworldwide.com, such as ns1a.ipamworldwide.com, for example. Each NSEC RRSet is also signed with the private ZSK, which in turn is validated against a trusted KSK.

NSEC3 provides similar authenticated denial of existence indication for RRSets, but it also obfuscates trivial enumeration of RRSets in the zone, which can be considered an information security risk. Because the NSEC3 record uses hashed owner names along with a salt value, which further complicates the hash dictionary creation function, it is much more computationally expensive to enumerate the zone. As such it is also computationally expensive when signing the zone and when validating query resolutions indicating record nonexistence.

13.4.3 Parent Delegation in a Chain of Trust

Now let's expand our example to illustrate the role of the DS record in generating an inter-zone chain of trust to a trust anchor. Consider Figure 13.3, where we will work our way up from the resolved data to the trust anchor. Let's assume the public KSK of the ipamworldwide.com. zone (key id = 06082) is configured as a trust anchor in my recursive server. When I issue an A record query for host.child.ipamworldwide.com., name resolution follows the traditional domain tree traversal to obtain a cached or authoritative answer. Assuming I had set the DO bit, the resolution RRSet along with the

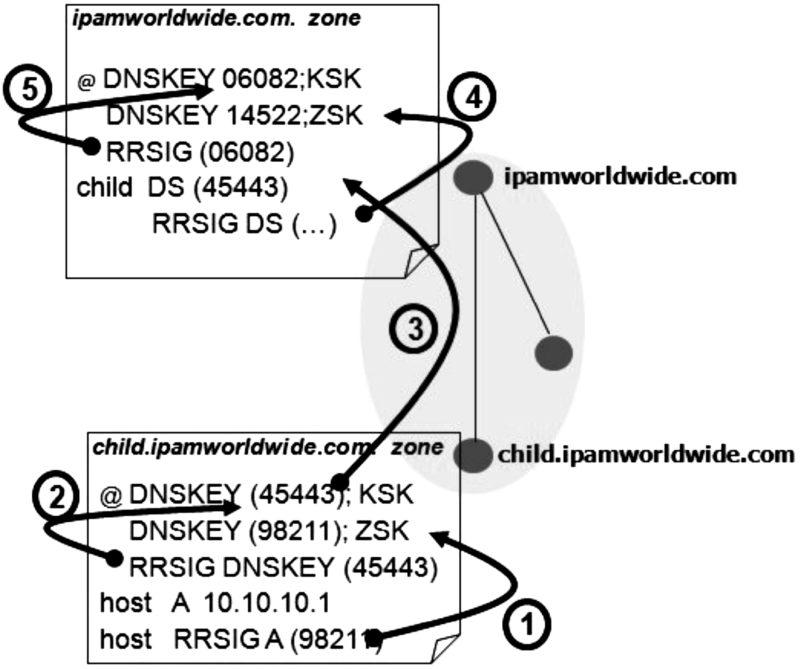

Figure 13.3. DNSSEC chain of trust traversal.

corresponding RRSIG record will be returned. The recursive server can validate the RRSIG with the child.ipamworldwide.com's ZSK (key id = 98211), shown as the arrow labeled "1" in Figure 13.3. In turn the ZSK can be validated against the zone's KSK (key id = 45443), per step 2. Given that I do not have this KSK configured as a trust anchor, I cannot trust this data. However, the recursive server queries the parent zone, ipamworldwide.com., for a DS record to determine if the parent can authenticate this zone's data. This is shown as step 3 in the figure.

If the DS record digest matches the corresponding child.ipamworldwide.com zone's KSK DNSKEY data, I can conclude that the ipamworldwide.com. zone has signed the delegation to child.ipamworldwide.com. The recursive server then validates the signature on the DS record against ipamworldwide.com's ZSK (key id = 14522) and in turn its KSK (key id = 06082), which is configured as a trusted key. Therefore, I have confirmed the original data resolved as trusted via the chain of trust back to a configured trust anchor! This same process could be repeated for any number of parent–child iterations up the domain tree to the signed root using the root zone trust anchor. I ultimately must have a trusted key configured for the zone or one of its ancestor zones within which my query applies. Considering the wide variety of zones for which queries need to be authenticated, including reverse zones, this set of trust anchors could quickly become very large!

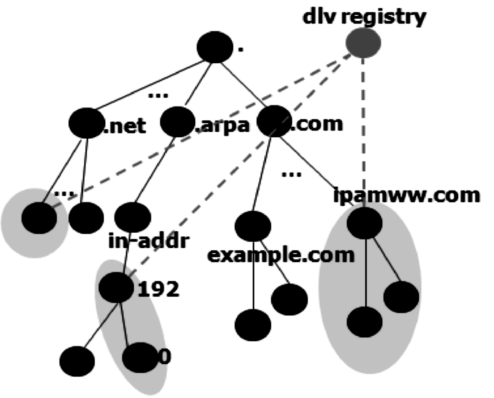

Figure 13.4. DLV chain of trust.

DNSSEC Lookaside Validation was defined to help keep the set of trust anchors to a manageable level in the time before the root zone was signed. DLV utilizes a centralized registry of signed zone public keys. By configuring the DLV registry as a trust anchor, you thereby trust the DLV registry and all "child" zones to which it authenticates. These zones are not actual child zones of the DLV, but are zones that the DLV authenticates. Zone administrators may register their signed zone keys with the DLV registry in a secure manner to maintain this "lookaside" or "sideways" chain of trust, as opposed the domain tree parent–child chain of trust we just discussed.

Figure 13.4 illustrates this concept. Without root and TLD zone signing, trusted keys had to be configured for each trusted zone. In the figure, these are illustrated as the ipamww.com, 192.in-addr.arpa and a .net zones. By using the DLV concept, the DLV signs the DS-equivalent DLV record to authenticate the KSK of the each "child" zone. The benefit of a DLV registry is to reduce the number of keys being managed in each recursive server in your organization. If the DLV signs three zone keys as illustrated in Figure 13.4, you need only be concerned with the DLV's key rollover, not the constituent three keys'. The DLV registry must be trusted fully, as zones for which it authenticates are not selectively accepted by registry users.

A trusted keys statement must be entered for the DLV registry and only one DLV registry may be so referenced. The DLV is identified in the recursive name server by configuring its public KSK in the `trusted-keys` statement block as we illustrated earlier. When building a chain of trust during name resolution, the recursive server will attempt to build the chain back to a configured trust anchor; should a valid chain not exist, it will attempt to validate the chain of trust through the DLV. The DLV registry must be able to authenticate its registered zones, much like the manner in which a parent zone validates its children's KSKs. The DLV resource record is used for this purpose, and its format is identical to the DS resource record type. It performs an equivalent function, though not in the traditional parent–child delegation chain.

Owner	TTL	Class	Type	RData			
DLV domain	TTL	IN	DLV	Key tag	Algorithm	Type	Digest
ipamww.com.dlv_reg.net.	86400	IN	DLV	32284	5	1	90df80DF89lLe. . ..

When the recursive server issues its last resort attempt to validate data in a zone, it seeks the DLV record corresponding to that zone in question within the DLV registry. The DLV registry is identified in the recursive server via the dnssec-lookaside statement, configured within the options block of named.conf. This statement identifies the branch of the domain tree for which escalations to the DLV registry are valid, as well as a reference to the trust anchor identified in the trusted-keys statement. For example, the statement below indicates that resolutions within the gov. domain could be escalated to dlv.us. and trusted as long as the dlv.us public KSK matches the configured dlv.us trusted key.

```
dnssec-lookaside ''gov'' trust-anchor ''dlv.us'';
```

13.5 KEY ROLLOVER

The most administratively intensive task with DNSSEC deals with the process of key rollover, particularly KSK rollover. Like passwords, keys must be periodically changed. It's best to provide a moving target to would-be attackers! The use of separate key-signing versus zone-signing keys helps the administration of this process. This is due to the fact that any zone administrator can simply resign his/her zone using a ZSK without affecting anyone else. Whatever ZSK is used, it is ultimately signed by the KSK, which may be configured as a trust anchor or referenced by a DS or DLV resource record. Thus, ZSKs can be changed at will. However, because KSKs are configured as trust anchors and are potentially referenced by other zones' DS or DLV records, they do impact other administrators and require a fairly tight integration process.

The two basic methods used for key rollover are preseeding the key, which is effective for rolling over ZSKs and the dual-key signature approach, which can be used to rollover KSKs. The key issue (no pun intended) with rollover relates to the updating of cached resolution and signature information in recursive servers and resolvers. When a resolver obtains authenticated resolution information, it will cache this information, including the DNSKEY records containing ZSKs and KSKs, for the duration of the original record TTL. After the TTL expires, the resolver must issue a new query for the corresponding information. If a zone administrator performs a flash cut of a new key for an old key, resolvers and recursive servers having performed queries with the old key, still valid per its TTL, will be unable to authenticate data resolved within the zone that was signed with the new key. This time impact is illustrated in Figure 13.5. Therefore, maintaining a window during which key updates can be propagated and two or more keys are valid comprise the basic rollover techniques.

Let's discuss and compare the two common rollover strategies by first considering Figure 13.6. Note that ZSK and KSK rollover should occur independently of one another

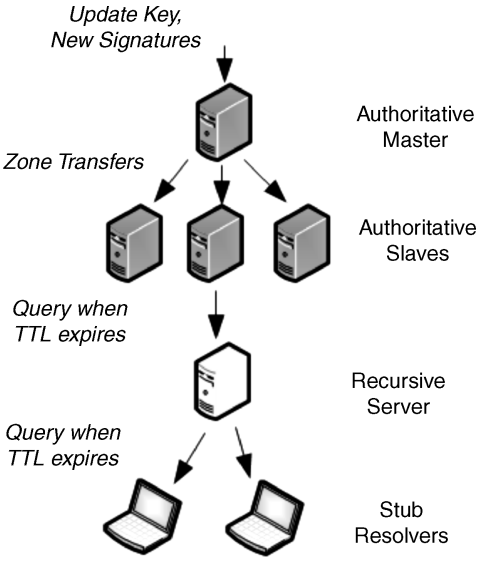

Figure 13.5. Zone information propagation.

if possible. We'll assume the preseed strategy is applied to ZSK rollover, while the dual-signature approach is applied to KSK rollover. Examining first ZSK rollover, our initial condition features a zone signed with ZSK [key tag] 14522 and KSK 6082 as indicated by the pen icon. At time t_0, the preseed time, a second, "passive" ZSK, 28004, is created using the dnssec-keygen utility or via BIND 9.7+ automation and its corresponding DNSKEY resource record is included in the zone file along with the active ZSK 14522. After ZSK 28004 has been inserted or included in the zone file, the zone must be resigned, still using ZSK 14522. The passive ZSK is itself signed by the active keys and made available for resolver and recursive server caching, but is not yet used to sign zone data.

Once published, both keys should remain in the zone file until all slave servers obtain the zone file via zone transfers plus the key expiration time. The key expiration time should be longer than the zone or resource record TTLs. When this time has passed at time t_1, the rollover time, the zone can be resigned, this time using the now formerly passive ZSK 28004. The formerly active ZSK can be left in the zone for the equivalent interval until time t_2, then removed from the zone file. Depending on the frequency of

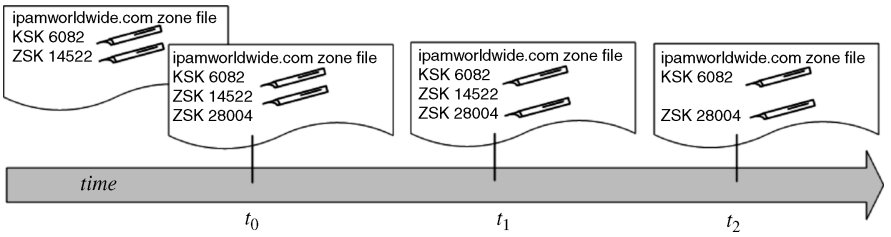

Figure 13.6. DNSSEC preseed key rollover strategy (11).

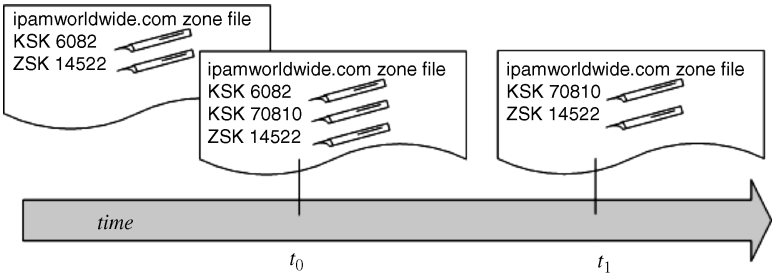

Figure 13.7. DNSSEC dual-signature key rollover strategy (11).

ZSK rollover, the time t_2 may correspond to t_0 in the next key rollover cycle, where the zone would always have two ZSKs, one active and one passive. Otherwise, at this point only the active ZSK exists within the zone.

Now let's examine the dual-signature rollover method, shown in Figure 13.7. This process starts with the same initial conditions as in our prior example. At rollover time, t_0, a new KSK (70810) is created using the dnssec-keygen utility with the –k option. Now sign the zone using the dnssec-signzone utility using *both* the current and new KSK and the active ZSK. Recall that dnssec-signzone permits specification of multiple KSKs. The public key of the new KSK must then be communicated to all resolvers/recursive servers that utilize this zone as a trust anchor. In addition, the parent zone that authenticates this zone must be updated.

An output of the dnssec-signzone utility when using the –g option includes a dsset-<zonename> file containing the corresponding DS resource record(s) that can be included in the parent's zone file. The –l option creates a dlvset-<zonename> file containing the corresponding DLV resource records. The parent zone or DLV administrator must copy or include these DS or DLV records, respectively and resign the parent zone. Given the manual configuration required to perform these tasks on the parent zone and resolvers/recursive servers, this time frame is less deterministic than the preseed method. Once this time has elapsed and the parent zone and trust anchor configurations have been updated, the old KSK may be removed from the zone file and the zone resigned using only the newly current KSK as shown at time t_1.

Emergency rollover procedures should be devised in the event of compromise of a private key corresponding to an active KSK or ZSK. Should an attacker obtain the private key, he/she could forge zone data and sign it with the private key. Resolvers and recursive servers would authenticate the falsified data based on the corresponding published public key. As we've seen, the ZSK can be changed autocratically and should be changed immediately. Changing the KSK however, does require broader involvement and coordination. We recommend documenting a process for emergency rollovers that includes the parent zone administrator and DLV registry contacts, as well as a means to communicate to users who have configured the KSK as a trust anchor. This could be via a registered email list and secure web site posting.

One other aspect of key updating is algorithm rollover. This involves the use of a new key-generation algorithm, for example, as a result of an algorithm compromise or

upgrade. As with changing the key itself, the dual signature process described above can be used to generate keys using new algorithms and roll them into production.

13.5.1 Automated Trust Anchor Rollover

RFC 5011 (143) defines a means to automate trust anchor rollover to reduce the administrative impact on updating trusted keys on all resolvers/recursive servers that utilize this zone as a trust anchor. This automation requires an initial configuration of the current public KSK for the trust anchor zone. But unlike manual configuration, it need not be manually updated every time the trust anchor KSK is changed. Automated trust anchor updates can be configured in a recursive BIND server using the `managed-keys` statement in BIND 9.7 and above. The initial trusted key will be used to validate future key transactions communicated using the DNS protocol. The resolver must periodically query the trust anchor for its DNSKEY RRSet to check for updates. If a new key is added correctly, it will be automatically considered a valid trust anchor key for the zone if signed by the current trusted key. If the current trusted key is revoked and is signed by the zone's trusted key(s), the trusted key will automatically be removed from processing. Hence the initial-key configured in the `managed-keys` statement is used as the trust anchor initial condition only; this key may be revoked in the future, and the DNS server automatically tracks the status of current trusted keys.

Figure 13.8 provides a state diagram of trusted keys from the resolver's perspective based on key states retrieved via validated (signed by current trusted key(s)) DNSKEY queries. When a new SEP key (trust anchor) is retrieved by the server within the DNSKEY RRSet, the key enters the Add Pending state. This state helps mitigate the case where an attacker has compromised the trusted key and seeks to convince the resolver to use the attacker's new key. If the resolver does not see the pending key in the DNSKEY

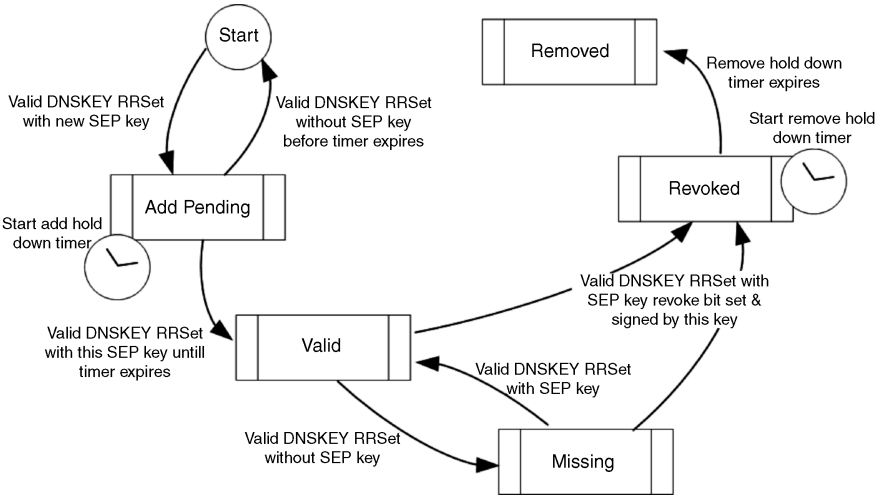

Figure 13.8. Trust anchor (SEP) state diagram (143).

RRSet at any time during the duration of the *add hold down* timer, the key will be considered invalid. It would be very challenging for an attacker to correctly respond to every DNSKEY query during this interval. Once the hold down timer expires, the trusted key enters the Valid state and is considered a valid trusted key for the zone. In this state if the key is missing from the DNSKEY RRSet, it will be considered Missing but will be reinstated to Valid state upon reappearance with valid signatures.

When the zone administrator desires to revoke the key due to its age in its lifecycle or because it was compromised, the key will be published in the zone with the Revoke bit set in the DNSKEY Flags field. This key must be used to sign the DNSKEY RRSet in addition to any other active or pending trusted keys. In this case, the key will be considered Revoked. This state may be entered from either the Valid or Missing states. The server begins a *remove hold down* timer, which upon expiration, stimulates removal of the trusted key from the server configuration.

13.5.2 DNSSEC and Dynamic Updates

You may be wondering given that the zone signing process requires the canonical ordering of a zone, then signature, how does one insert a new resource record into the zone securely? Fortunately, zone signing does not require reprocessing of the entire zone. Individual RRSets are signed, enabling a more modular process. However, the NSEC[3] records must be adjusted to account for the update, effectively inserting the update by adjusting the canonical ordering.

When dyamically updating a secure zone, the update itself must be secure. The server should require signatures on update messages and should define which servers or networks may perform updates. When an update has been received and authenticated, it remains within the journal file. To fully sign the zone with the update, the server must temporarily freeze dynamic updates, when using pre-BIND 9.6, via the `rndc freeze` command. This shuts off acceptance of dynamic updates. Once frozen, the zone must be resigned using the `dnssec-signzone` functionality. Then, dynamic updates may be reenabled using the `rndc thaw` command.

This manual freeze-sign-thaw process has been obviated in BIND 9.6 and above which has added an automated signing mechanism for dynamic updates, vastly simplifying this process. Along with its normal integration of journal updates into the zone file, BIND signs each update using the ZSK along with the corresponding "before" and "after" NSEC[3] records to canoncially insert the record into the zone. BIND 9.6 also periodically examines the zone for signatures nearing expiration. It will then automatically generate new signatures in such cases. To perform this automated signature process, BIND must have access to the ZSK private key to sign or resign records.

13.5.3 DNSSEC Deployment Considerations

BIND provides several utilities for the creation of keys and for signing zones to simplify the DNSSEC implementation process. However, consider the following carefully when deciding to deploy DNSSEC

- Decide which zones you want to sign. In general, the first and perhaps only zones to consider signing are your public or external zones. These enables users resolving your public name space to do so securely and reduces the probability of an attacker "impersonating" your zone. Partner connections via the Internet should likewise be considered. Otherwise, internal resolution of internal information can usually be trusted within most organizations.
- As we've seen the size of a signed zone file is much larger than that of a corresponding unsigned zone file. This may affect required server memory for large zones as well as zone loading time.
- Signing your zones protects the integrity of your name space but not your DNS caches. Consider configuring DNSSEC validation on your Internet-querying DNS servers.
- The resolution response for a given query would also grow larger, given the attachment of RRSIG records and potentially DNSKEY records corresponding to the query. This could adversely affect query response time and performance.
- The resolution process performance may also be further adversely impacted by the trust anchor confirmation process, where keys and delegation signer records are validated up to the trust anchor zone or DLV registry.
- DNSSEC introduces the requirement for time synchronization given the absolute time references denoting valid and expiration times in RRSIG records
- Zone footprinting by hopping NSEC records is a potential information overexposure though the NSEC3 record makes this process more difficult. Consider whether zone footprinting is really an issue for you (generally information published in DNS is public information!) due to the computational complexity and potentially time for generating NSEC3 records within a signed zone.
- Key update procedures for initialization and rollover must be devised to provide authenticated access to updated KSKs via an out of band mechanism if your trusting resolvers are not using the automated trust anchor update feature. The KSK public key update must be communicated to all who trust your zone as well as your parent zone or DLV, if any, in the form of a DS or DLV record, respectively.
- DNSSEC performs data origin authentication, data integrity verification, and authenticated denial of existence. It does not protect against other vulnerability types introduced in Chapter 12. Don't forget to implement the mitigation tactics discussed in that chapter to protect against other vulnerabilities.