A Practical Crash Course Intro to DANE SMIME Implementation

Objective: To convey, as rapidly as possible, what's needed for practical implementation of DANE SMIME. Background is omitted: goal is to get working examples and code.

DANE SMIMEA is based on DANE TLSA. So, let's first examine DANE TLSA.

TLSA Example

To get the TLSA for good.dane.verisignlabs.com (for 443/tcp), query: *(Color coding has been added)*

```
$ dig +dnssec +noall +answer +multi _443._tcp.good.dane.verisignlabs.com. -t
TYPE52
_443._tcp.good.dane.verisignlabs.com. 60 IN TYPE52 \# 35 (
030001 0332AA2D58B3E0544B65656438937068BA44CE
                                2F14469C4F50C9CC6933C808D3 )
```

Note: On newer versions of dig, you can do specify "TLSA" instead of "-t TYPE52", and it will parse the results a bit better. Example from http://blog.huque.com/2012/10/dnssec-and-certificates.html :

```
$ dig +dnssec +noall +answer +multi _443._tcp.www.huque.com. TLSA
_443._tcp.www.huque.com. 893 IN TLSA 3 0 1 (
8CB0FC6C527506A053F4F14C8464BEBBD6DEDE2738D1
                                1468DD953D7D6A3021F1 )
```

Now, you'll see there are 4 fields:

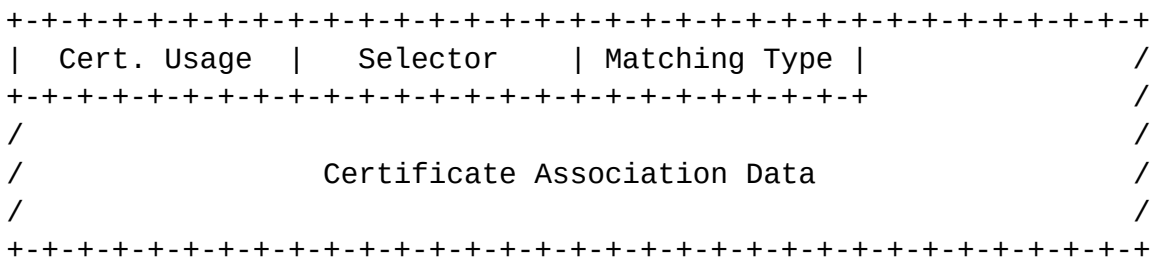
1. First octet is "Certificate Usage" – in this case, "**3**"
2. Second is "Selector" – in this case, "**0**"
3. Third is "Matching Type" – in this case, "**1**"
4. Fourth (and final, until end of RDATA) is "Certificate Association Data" – in this case, "8CB0FC…"

What do the fields mean?

For most SMIMEA cases, we're only concerned with the 3rd and 4th fields. But let's understand each one, regardless. Fields are defined in length in http://tools.ietf.org/html/rfc6698#section-2.1 , and explained more simply in http://blog.huque.com/2012/10/dnssec-and-certificates.html and https://singapore49.icann.org/en/schedule/wed-dnssec/presentation-dnssec-dane-26mar14-en.pdf .

In short, the RDATA looks like:

```
                  1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```



The fields mean:

- **Certificate Usage**
  - 0 = “CA constraint”
  - 1 = “service certificate constraint”
  - 2 = “trust anchor assertion”
  - 3 = “domain-issued certificate”
- **Selector**
  - 0 = full certificate
  - 1 = public key only
- **Matching Type**
  - 0 = exact match
  - 1 = match SHA-256 hash
  - 2 = match SHA-512 hash
- **Certificate Association Data**
  - full certificate, or public key data, or hash value

Cert Usage 0 and 2 mean the record specifies a CA that the TLS cert must match, but doesn’t give the cert itself. This is not very useful for SMIMEA. Cert Usage 1 and 3 mean we are giving the cert (1 means it will be signed, 3 means the fact that it is in DANE is good enough. ***Question: Does that mean we need to verify the cert using CA if it is Cert Usage 1?***)

Selector 0 means we are giving a cert; 1 means we are giving only a bare key. This is imporant for how we parse the actual data (fourth field).

Matching Type 1 or 2 means we are only giving a hash of the cert/key. This is not very useful for SMIMEA. For SMIMEA, expect Matching Type 0.

Finally, Certificate Association Data is the actual cert or bare key.

What does this mean for SMIMEA?

Certificate Usage must be 1 or 3, Matching Type must be 0. If Selector is 0, fourth field is S/MIME cert. If Selector is 1, fourth field is S/MIME bare key. SMIMEA will be discussed more later; for now, we focus on TLSA.

How can I do the DNS query in code?

Obviously, you can't just use gethostbyname() or the equivalent, since we want a TLSA record, not an A record with an IP address. So we need to use a DNS library to a) specifiy the record type (in this case, TLSA, or 52) and b) get the full RDATA back (not just a parsed IP address).

Easy. Using dnsjava (from `xbill.org`) and Scala, it's:

```
scala> import org.xbill.DNS._
import org.xbill.DNS._

scala> val r = new Lookup("_443._tcp.good.dane.verisignlabs.com",
Type.TLSA).run()(0)
r: org.xbill.DNS.Record = _443._tcp.good.dane.verisignlabs.com. 60      IN
TLSA    3 0 1
0332AA2D58B3E0544B65656438937068BA44CE2F14469C4F50C9CC6933C808D3
```

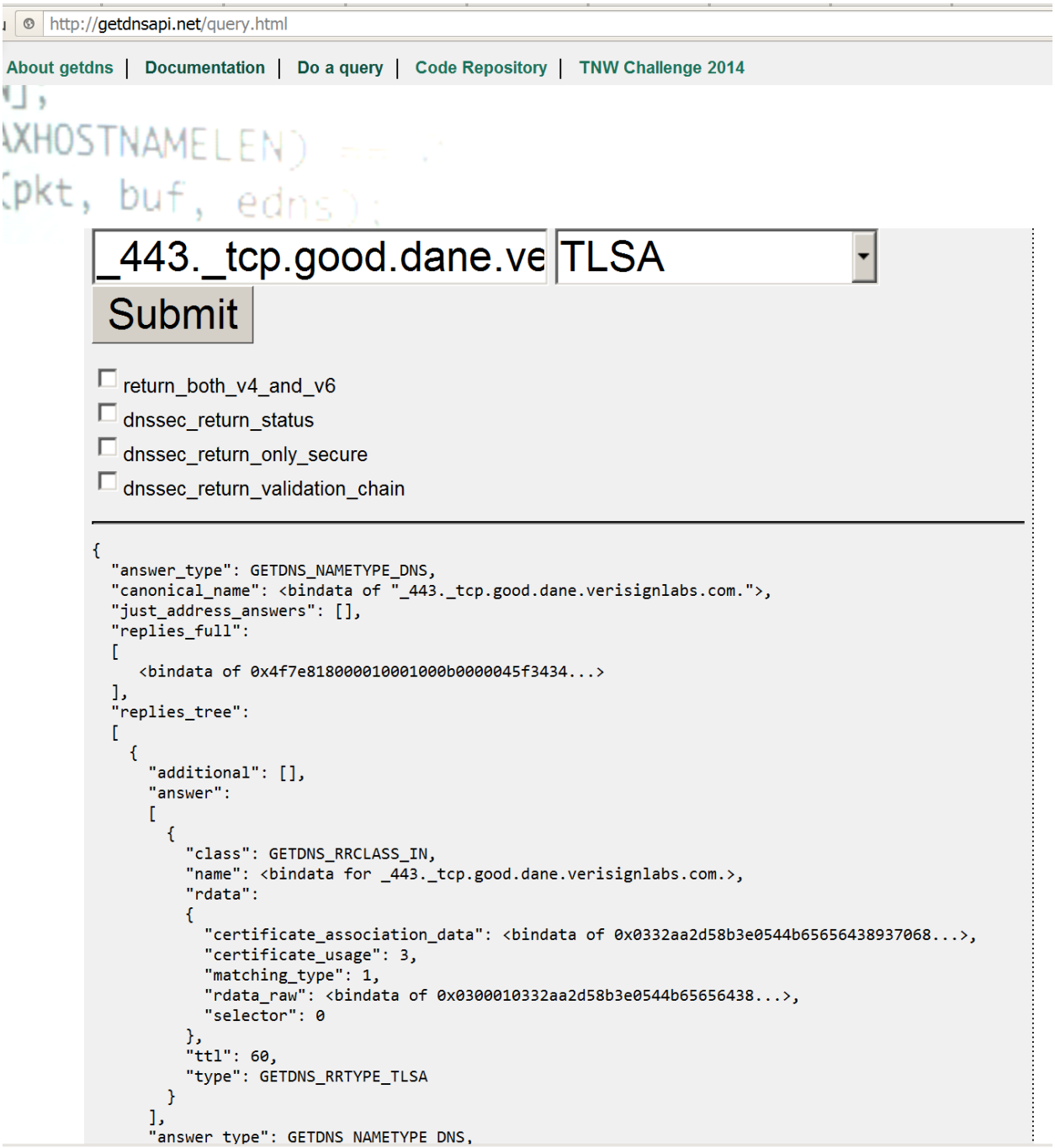
`Lookup` can return multiple records; all of those should be treated as valid. But, for simplicity, we just looked at the first (by adding the "`(0)`" at the end).

You can easily read the four fields directly out of the returned `rdataToString`, but, if you want, you can have dnsjava do it for you:

```
scala> val tlsa = r.asInstanceOf[TLSARecord]
tlsa: org.xbill.DNS.TLSARecord = _443._tcp.good.dane.verisignlabs.com.  60
IN      TLSA    3 0 1
0332AA2D58B3E0544B65656438937068BA44CE2F14469C4F50C9CC6933C808D3

scala> tlsa.getCertificateUsage
res14: Int = 3

scala> tlsa.getSelector
res15: Int = 0

scala> tlsa.getMatchingType
res16: Int = 1

scala> tlsa.getCertificateAssociationData.map("%02X" format _).mkString
res17: String =
0332AA2D58B3E0544B65656438937068BA44CE2F14469C4F50C9CC6933C808D3
```

Note: The above example relies on the system recursive resolver, and, importantly, didn't do any DNSSEC validation. As discussed elsewhere, DNSSEC validation must be done; it's a design decision whether to do it in the app, the OS, or the network's resolver.

Using getdns API, it's even easier:



Okay, I understand TLSA. What about SMIMEA?

SMIMEA is not that much different. The differences are:

1. IANA has not yet assigned a RRtype for SMIMEA. So, for now, just use a “private use” one (similar to RFC1918 IP addresses), that is, a vallue between 65280-65534 (see http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4).

2. Instead of prefixing the domain with "_443._tcp", hash and prefix the email address: robert@bigbank.com becomes hex(ssh-224(robert))._smimecert.bigbank.com, or:

`ecb453d83da2067efeb1243964d2b00aca62a7230c64e39ce69ac555._smimecert.bigbank.com`.

SHA-224 support should be added to Java 8 (see https://blogs.oracle.com/mullan/entry/jep_130_sha_224_message) and seems to be already supported in Bouncy Castle (http://www.cs.berkeley.edu/~jonah/bc/org/bouncycastle/crypto/digests/SHA224Digest.html).

Question: How do we handle case of the email address, when hashing? Do we just lowercase everything?

3. You may get multiple responses; each one is valid (eg someone has multiple certs for different devices)

4. The fourth field, Certificate Association Data, contains the cert (or bare key?) **TODO: Parsing this and feeding the cert or key to the MUA.**

How does this fit in to the MUA?

TODO