CertID-KeyID

(and other issues)

Syntaxes for Unambiguous Identification of Certificates and Public Keys

(Sean Leonard, Penango, Inc.)

IETF 77, 2010-03-22

draft-ietf-pkix-certid-keyid-01

(Not PKIX WG (yet), sorry)

Misunderstandings

GeneralName != Name

GeneralName != Identity

GeneralName != Identify

GeneralName != Authentication/A'zn

GeneralName == PROTOCOL ELEMENT

- A way to represent data (sometimes, not always, *identifiers*) in PKIX
- How to use this data is context-dependent

Problem

How to identify another certificate and a key unambiguously…

…in a GeneralName?

- Wrong question
- GeneralName == “everything EXCEPT other certificates and keys”

Technical Problem

Given the tools we have, how can we: safely, securely, simply, unambiguously, and uniformly…

identify a certificate (or key) in PKIX or application-specific protocols?

- Using the same method(s) and the same code paths, *because it's the same problem.*
- Standards Track…or BCP

Existing Cert IDs

ASN.1	RFC	ASN.1	RFC
ESSCertID	2634	ESSCertIDv2	5035
CertID/OCSP	2560	SCVPCertID	5055

- Possibly more (haven't reviewed everything)
- Can we just have one please?

Candidate: ESSCertIDv2

```
ESSCertIDv2 ::=  SEQUENCE {
        hashAlgorithm          AlgorithmIdentifier
             DEFAULT {algorithm id-sha256},
        certHash               Hash,
        issuerSerial           IssuerSerial OPTIONAL      }
    Hash ::= OCTET STRING
    IssuerSerial ::= SEQUENCE {
        issuer                 GeneralNames,
        serialNumber           CertificateSerialNumber      }
```

PKIXCertID ::= ESSCertIDv2

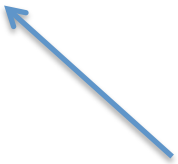
(ASN.1 Module optional; can just be guidance to authors)

Keys

- Why????
- Same principles
- Same problem
- Same solution
- PKIX already does it
  (just doesn't want to admit it ☺)

Keys

```
ObjectDigestInfo ::= SEQUENCE {
    digestedObjectType  ENUMERATED {
     publicKey          (0),
     publicKeyCert      (1),
     otherObjectTypes   (2) },
    -- otherObjectTypes MUST NOT
    -- be used in this profile
    otherObjectTypeID   OBJECT IDENTIFIER OPTIONAL,
    digestAlgorithm     AlgorithmIdentifier,
    objectDigest        BIT STRING     }
```



Annoying, but it works

Keys by Value

SubjectPublicKeyInfo
…in certificate

No other PKIX-sanctioned way; certs or bust

Conclusions

- PKIX protocols/extensions “SHOULD” use these
- Application-specific protocols/extensions “MAY” use these…
  - But uniform tools mean uniform code to do it.
  - Safe, Secure, Unambiguous
  - Simple? (Close enough…)

END (of this issue)

Questions & Discussion


ZOO MED'S
CAN O' WORMS
Net Wt.
1.2 oz. (35g)

Patterns

- Do we want to talk about this?
- "A method of specifying and applying access control rules"…
  - By computers
  - For computer consumption
  - Not human consumption per-se (if you want that, see Subject name, draft-ietf-pkix-certimage, etc.)
- Least Privilege
  - Authority has authority over whole scope (all example.com), but voluntarily chooses to restrict scope to least privilege
  - Broader than single URI (http://foo.example.com/service), but lesser than whole DNS host (*://foo.example.com/*)

Problem

- Class of resources known, defined by URIs
- Interpretation of URIs very scheme-specific
- But all URIs have common format: they are all ASCII strings (or Unicode strings for IRIs)
- (Compare with BURLs [RFC 4468], IMAP AUTH URLs [RFC 5092])

Specific Use Case

- (Hopefully non-controversial)

- AC Targeting Extension, RFC 5755
- Specify (honest) services that MAY use the AC

```
Target  ::= CHOICE {
      targetName         [0] GeneralName,
      targetGroup        [1] GeneralName,
      targetCert         [2] TargetCert
    }
```

Specific Use Case

- Match foo.example.com/websockets/*
- * is invalid URI character
  - Use regular expressions
- URIs complicated to parse
  - Specify URI components
  - Assume URI parser (app has anyway)

URI->Path = /^\/websockets/

"WebSockets:" cf. http://dev.w3.org/html5/websockets/

END (of this issue)

Questions & Discussion