TWO SIGMA

OpenSSL needs an ASN.1 compiler

Nico Williams

00.00.00

What is ASN.1

- Abstract Syntax Notation 1: an IDL
- Also a family of encoding rules (ERs)
  - BER (and DER and CER) (tag-length-value)
  - OER (and PER) (XDR-like, no tags)
  - XER (XML!)
  - JER (JSON!)
  - GSER, etc.

What is ASN.1

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}
```

What is ASN.1

```
TBSCertificate  ::=  SEQUENCE  {
    version         [0]  EXPLICIT Version DEFAULT v1,
    serialNumber         CertificateSerialNumber,
    signature            AlgorithmIdentifier,
    issuer               Name,
    validity             Validity,
    subject              Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
    extensions      [3]  EXPLICIT Extensions OPTIONAL
}
```

What is ASN.1

- It’s ubiquitous
  - Kerberos
  - x.509/PKIX
  - lots of ITU-T and financial protocols
- Why not Protocol Buffers, amirite

What is ASN.1

- ASN.1 the syntax:
  - x.680 (base), x.681 (IOS), x.682 (constraints), x.683 (parameterization)
- ERs
  - X.690 (BER/DER/CER), x.691 (PER), x.693 (XER), x.696 (OER), x.697 (JER)

Hand-coding ASN.1 codecs is fraught

- There have been many critical vulnerabilities due to hand-coded codecs for things like XDR, ASN.1/DER, and so on. It's just repetitive, manual, and error prone.
  - Transcription errors suck
- AI won't save us: its hallucinations will be hard to find in a sea of repetitive generated code!

Benefits of having ASN.1 tooling

- Security
- Labor
- Speed of iteration
  - Support new features sooner, with less effort

ASN.1 C tooling is hard to come by

- Well, is that true? There are a whole bunch...
- But OpenSSL needs an open source tool that outputs unlicensed, unencumbered code, and it should be free and open source.
- That seems to leave these:
  - SNACC / eSNACC
  - asn1c
  - Heimdal's `asn1_compile`
  - Write your own (not *that* hard)

Moving to ASN.1 tooling is ETOOHARD?

- It would be nice to be able to keep OpenSSL's ASN.1 C machinery, at least for a while, no?
- Type codegen impedance mismatches
  - Codegen controls needed

Requirements

- ASN.1 (x.680)
- DER (x.690)
  - but not necessarily BER and definitely not CER
- A way to access `tbsCertificate` as it was received even after decoding `Certificate` values
- Data type codegen controls

What about RFCs 5911 & 5912?

- RFCs 5911 and 5912 are PKI that use the ASN.1 IOS to *formally* specify things that earlier RFCs specified in English *prose*
  - `Certificate`, CSR, and other `Extensions`
  - `OtherName`s
  - Algorithm IDs and parameters
  - `WITH SYNTAX` and `Signed{<Type>}`

What about RFCs 5911 & 5912?

```
-- RFC 5280
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}

-- RFC 5912
Certificate  ::=  SIGNED{TBSCertificate}
```

What about RFCs 5911 & 5912?

```
TBSCertificate  ::=  SEQUENCE  {
   version         [0]  Version DEFAULT v1,
   serialNumber         CertificateSerialNumber,
   signature            AlgorithmIdentifier{SIGNATURE-ALGORITHM,
                        {SignatureAlgorithms}},
   issuer               Name,
   validity             Validity,
   subject              Name,
   subjectPublicKeyInfo SubjectPublicKeyInfo,
   ... ,
   [[2:                 -- If present, version MUST be v2
   issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
   subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL
   ]],
   [[3:                 -- If present, version MUST be v3 --
   extensions      [3]  Extensions{{CertExtensions}} OPTIONAL
   ]], ... }
```

What about RFCs 5911 & 5912?

```
-- RFC 5280
Extension  ::=  SEQUENCE  {
    extnID      OBJECT IDENTIFIER,
    critical    BOOLEAN DEFAULT FALSE,
    extnValue   OCTET STRING
                -- contains the DER encoding of an ASN.1 value
                -- corresponding to the extension type identified
                -- by extnID
}
```

What about RFCs 5911 & 5912?

- What can we do with RFCs 5911 and 5912?
  - Automatic, fully recursive, one-shot encoding and decoding of certificates and CSRs, alg ID & params correspondence validation, and whatnot!

What about RFCs 5911 & 5912?

```
struct Extension {
  heim_oid extnID;
  int critical;
  heim_octet_string extnValue;
  struct {
  enum { choice_Extension_iosnumunknown = 0,
         choice_Extension_iosnum_id_pkix_pe_authorityInfoAccess,
         ... } element;
  union { void *_unknown;
          AuthorityInfoAccessSyntax *ext_AuthorityInfoAccess;
          ... } u;
  } _ioschoice_extnValue;
}
```

What about RFCs 5911 & 5912?

```
: git:heimdal[(HEAD detached at origin/master)]:TOP%; build/lib/asn1/asn1_print lib/asn1/fuzz-inputs/minimal-ek.crt Certificate | jq -C . | head -20
Match: Certificate
{
  "_type": "Certificate",
  "tbsCertificate": {
    "_type": "TBSCertificate",
    "_save": "30820265A00302010202146A0597BA71D7E6D3AC0EDC9EDC95A15B998DE40A300D06092A864886F70D01010B05003055310B300906035504061302434831E301C060355040A
3550403131D53544D2054504D20454B20496E7465726D656469617465204341203035301E170D313831323134303030303030305A170D323831323134303030303030305A30003013300D06092A864
183016801414ADB994AB58BE57A0CC9B900E7851E1A43C0866030420603551D20043B303930370604551D2000302F302D06082B06010505070201162168747470342F2F7777772E73742E636F6D
04DA44B3049311630140605678105020010C0B69643A353335343444323031173015060567810502020C0C535433334854504841484330311630140605678105020300C0B69643A3030343930303
2E3002010002020008A3043060567810502123133A30380201000101FFA0030A0101A1030A0100A2030A0100A310300E1603332E310A01040A01020101FFA40F300D16053134302D320A01020101
2300030100603551D25040930070605678105080130 4A06082B06010505070101043E303C303A06082B06010505073002862E687474703A2F2F7365637572652E676C6F62616C7369676E2E636
    "version": "rfc3280_version_3",
    "serialNumber": "6A0597BA71D7E6D3AC0EDC9EDC95A15B998DE40A",
    "signature": {
      "_type": "AlgorithmIdentifier",
      "algorithm": {
        "_type": "OBJECT IDENTIFIER",
        "oid": "1.2.840.113549.1.1.11",
        "components": [
          1,
          2,
          840,
          113549,
          1,
          1,
          11
: git:heimdal[(HEAD detached at origin/master)]:TOP%;
```

What about RFCs 5911 & 5912?

```
: git:heimdal[(HEAD detached at origin/master)]:TOP%; build/lib/asn1/asn1_print /tmp/ee-self-signed.der Certificate|jq '.tbsCertificate.extensions[] | select(.extnID.name=="id-x509-ce-subjectAltName")'
Match: Certificate
{
  "_type": "Extension",
  "extnID": {
    "_type": "OBJECT IDENTIFIER",
    "oid": "2.5.29.17",
    "components": [
      2,
      5,
      29,
      17
    ],
    "name": "id-x509-ce-subjectAltName"
  },
  "critical": true,
  "extnValue": "3011810F666F6F406261722E6578616D706C65",
  "_extnValue_choice": "ext-SubjectAltName",
  "_extnValue": [
    {
      "_choice": "rfc822Name",
      "value": "foo@bar.example"
    }
  ]
}
: git:heimdal[(HEAD detached at origin/master)]:TOP%;
```

Whetting your appetite

- What if we can support codegen of modules that use OpenSSL’s ASN.1 C macros?

Whetting your appetite

- What if we can support codegen of modules that use OpenSSL's ASN.1 C macros?
- Heimdal's `asn1_compile` can output a JSON representation of an ASN.1 module that can be used to drive codegen however you want
- One could even use `jq` to drive codegen based on this!

`--preserve-type=TBSCertificate`

```
struct TBSCertificate {
  heim_octet_string _save; // <-- saves the DER encoding of
                           //     this at decode time
  Version *version;
  CertificateSerialNumber serialNumber;
  AlgorithmIdentifier signature;
  Name issuer;
  ...
};
```

`--preserve-type=TBSCertificate`

- Simplifies signature validation
  - No need to re-encode
  - DER would never have been needed had this been popularized 40 years ago

`--decorate=<decoration-spec>`

- Add fields for intrusive data structures to generated C structs!
  - --decorate=ASN1-TYPE:FIELD-ASN1-TYPE:fname
  - --decorate=ASN1-TYPE:void*:fname
  - --decorate=ASN1-TYPE:FIELD-C-TYPE:fname[?]:[copyfn]:[freefn]:header

- You can decorate with another ASN.1 type, or with a host language (C) type
  - Copy constructors and destructors will work as expected

Options

- Neither asn1c nor SNACC / eSNACC support things like Heimdal's `--preserve-binary=TYPE` option
- Heimdal and asn1c have some support for RFCs 5911 / 5912

Parsing ASN.1

- Parsing ASN.1 is tricky because it has some ambiguities that can only be resolved when the module has been fully parsed
  - Values vs. objects
  - Types vs. object sets
  - Forward declarations not needed
    - Need to parse the whole module first, then resolve

Parsing ASN.1

- `WITH SYNTAX` is beyond LALR(1) parser generators
- On the plus side, LLMs are getting really good at codegen… Writing an ASN.1 compiler is getting easier.
- And if the goal is to produce the macro-using modules that OpenSSL has now, then codegen should be quite easy

Parsing ASN.1

- At least ASN.1 can be parsed with an LL(k) parser generator
  - Or a left-recursive parser with backtracking

C data type codegen controls

- In-band controls require editing ASN.1 modules
  - e.g., size constraints on INTEGER → pick smallest int type that fits
- Out-of-band controls require addressing by type and member name
  - e.g., DEFAULT members made pointers or not

C data type codegen controls

- Choices of base C types for base ASN.1 types
  - int sizes
  - string types
  - REAL type(s) (not needed for PKIX though)
  - etc.

Codec codegen styles

- Heimdal has two compiler backends
  - Codegen – function per type
  - “template” – think byte-coding
- Codec perf might not matter in a crypto-heavy library
  - Still, template uses less memory, thus less i-cache, d-cache, bandwidth

Codec codegen styles (templates)

```
struct asn1_template {
    uint32_t tt;     // encodes tag and type info
    uint32_t offset; // offset into C struct of field
    const void *ptr; // pointer to sub-template or
                     // other relevant info
};
```

Codec codegen styles (templates)

- `tt` field has
  - 4 bits of opcode
  - 4 bits of flag
  - 1 bit to indicate the type of `ptr`
  - 2 bits of class
  - 21 bits of tag number
-

Codec codegen styles (templates)

- `struct template` could be further size-optimized, though maybe at a cost of many more branches
  - Currently templates for PKIX are 59% smaller than code

Practical ways forward

- Use Heimdal's compiler
  - Directly, or indirectly using its JSON output
- Write your own tooling?
  - Mimic Heimdal's?
- Fork Heimdal's?
  - License issues...

Practical ways forward

- Maybe Heimdal could donate its compiler?
  - Only 21 authors, fewer when excluding trivial contributions

Practical ways forward (if using Heimdal's)

- Use Heimdal's asn1_compile's JSON output to drive codegen?
- Use Heimdal's asn1_compile?
  - If using the "template" approach asn1_compile can generate templates and OpenSSL can generate C types and an libcrypto can provide template interpreters

https://github.com/heimdal/heimdal

- heimdal/heimdal
  - lib/asn1/