MTAT.07.017
Applied Cryptography

Abstract Syntax Notation One (ASN.1)

University of Tartu

Spring 2018

Abstract Syntax Notation One

Notation to describe *abstract* types and values

Describes *information* – not representation

Similar to XML schema, however:

- ASN.1 is rich with built-in data types
- ASN.1 is not tied to particular encoding mechanism

Most cryptography standards use ASN.1 to describe data protocols and storage formats

ASN.1 example

```
-- ASN.1 module
MyQAProtocol DEFINITIONS ::= BEGIN
   MyQuestion ::= SEQUENCE {
      id INTEGER (0..999),
      text UTF8String
   }

   MyAnswer ::= SEQUENCE {
      id INTEGER (0..999),
      text UTF8String
   }
   -- new type defined
END
```

ASN.1 simple types

```
NULL -- only possible value is Null
BOOLEAN -- True or False
INTEGER -- whole numbers -infinity..+infinity
REAL -- mantissa, base, exponent
OCTET STRING -- values 0x00..0xFF
BIT STRING -- 0-s and 1-s
UTF8String -- UTF-8 characters
NumericString -- [space]0123456789
PrintableString -- printable ASCII chars
IA5String -- ASCII chars 0x00..0x7F
BMPString -- UNICODE BMP code points
UTCTime -- time in form "YYMMDDhhmmssZ"
```

There are more...

ASN.1 structured types

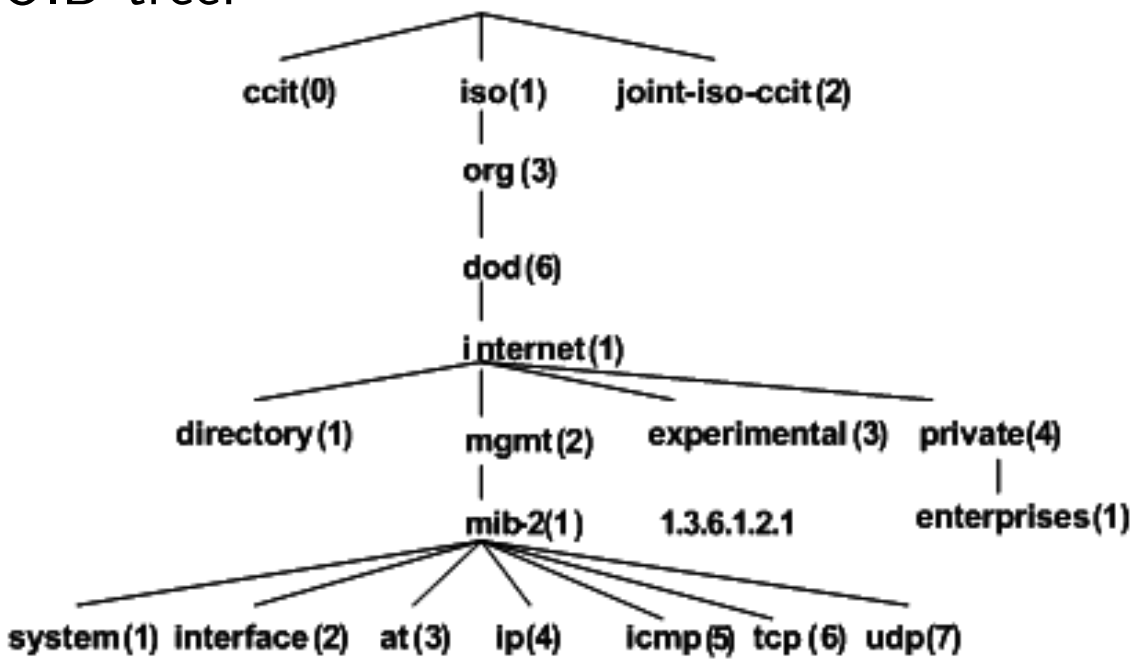
```
YearInfo ::= SEQUENCE {
    year         INTEGER (0..9999),
    isLeapYear BOOLEAN
}

Person ::= SET {
    name        IA5String,
    age         INTEGER,
    female      BOOLEAN
}

Prize ::= CHOICE {
    car          IA5String,
    cash         INTEGER,
    nothing      NULL
}
```

ASN.1 OBJECT IDENTIFIER

```
Algorithm  ::=  OBJECT IDENTIFIER
rsa  Algorithm ::=  {1.2.840.113549.1.1.1}
```

iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) rsaEncryption(1)

OID tree:



http://oid-info.com

ASN.1 encodings

```
-- ASN.1 type definition
Question ::= SEQUENCE {
    id INTEGER,
    questionText UTF8String
}
```

How do we encode this structure for transmission?

The standard ASN.1 encoding rules:

- Basic Encoding Rules (BER)
- Canonical Encoding Rules (CER)
- Distinguished Encoding Rules (DER)
- Packed Encoding Rules (PER)
- XML Encoding Rules (XER)
- Generic String Encoding Rules (GSER)

XML Encoding Rules (XER)

```
-- ASN.1 type definition
Question ::= SEQUENCE {
    id INTEGER,
    questionText UTF8String
}

<!-- XER-encoded object -->
<Question>
  <id>42</id>
  <questionText>Why is it so?</questionText>
</Question>
```

- Human readable
- Inefficient encoding
- Canonicalization needed

Distinguished Encoding Rules (DER)

- Efficient encoding
- A value can be encoded only in a single way
- Data is encoded as type-length-value (TLV) element:

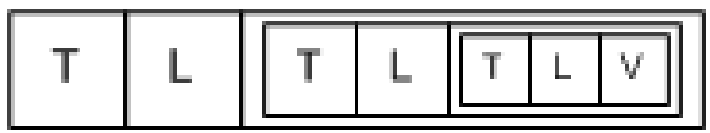
```
msg UTF8String :: = "Hello"
```

Type: UTF8String
Length: 5 bytes
Value: "Hello"

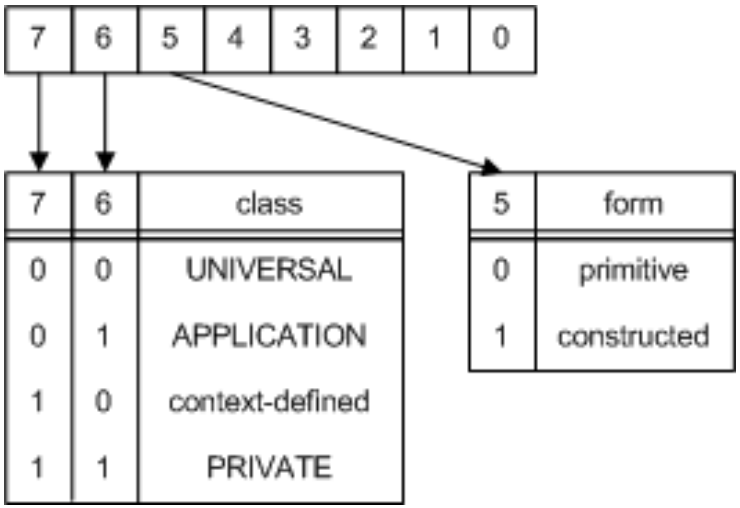
DER encoded:

```
[0x0c] [0x05] [0x48 0x65 0x6c 0x6c 0x6f] ...
```

```
$ echo -e -n "\x0c\x05Hello" > hello.der
$ sudo apt install dumpasn1
$ dumpasn1 hello.der
  0   5: UTF8String 'Hello'
```



Type-Length-Value: Type




```
Universal tags (Bits 4,3,2,1,0):
00001 (1) - BOOLEAN
00010 (2) - INTEGER
00011 (3) - BIT STRING
00100 (4) - OCTET STRING
00101 (5) - NULL
00110 (6) - OBJECT IDENTIFIER
01010 (10) - ENUMERATED
01100 (12) - UTF8String
10000 (16) - SEQUENCE
10001 (17) - SET
10011 (19) - PrintableString
10111 (23) - UTCTime
...
```

```
Class (Bits 7,6):
0 universal -- meaning is the same in all applications
1 application -- meaning is specific to an application
2 context-defined -- meaning is specific to a given strucutred type
3 private -- meaning is specific to a given enterprise
```

0x0c – 00 0 01100 (universal, primitive, UTF8String)

A Layman's Guide to a Subset of ASN.1, BER, and DER:

`http://luca.ntop.org/Teaching/Appunti/asn1.html`

ASN.1 encoding rules: Specification of BER, CER and DER:

`http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf`

Task: ASN.1 DER encoder

Implement ASN.1 DER encoder that can encode subset of ASN.1 types by implementing these functions:

```
def asn1_boolean(bool):
def asn1_integer(i):
def asn1_bitstring(bitstr):
def asn1_octetstring(octets):
def asn1_null():
def asn1_objectidentifier(oid):
def asn1_sequence(der):
def asn1_set(der):
def asn1_printablestring(string):
def asn1_utctime(time):
def asn1_tag_explicit(der, tag):
def asn1_len(content): <-- helper function
```

Task: ASN.1 DER encoder

And encodes this artificial ASN.1 structure (test case):

```
$ ./asn1_encoder.py asn1.der
$ dumpasn1 asn1.der
  0 114: [0] {                    explicit tags
  2 112:   SEQUENCE {
  4  16:     SET {
  6   1:       INTEGER 5
  9   4:       [2] {
 11   2:         INTEGER 200
      :         }
 15   5:       [11] {
 17   3:         INTEGER 65407
      :         }
      :       }
 22   1:     BOOLEAN TRUE
 25   2:     BIT STRING 5 unused bits
      :       '011'B
 29  51:     OCTET STRING
      :       00 01 02 02 02 02 02 02 02 02 02 02 02 02 02 02
      :       02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
      :       02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
      :       02 02 02
 82   0:     NULL
 84   7:     OBJECT IDENTIFIER '1 2 840 113549 1'
 93   6:     PrintableString 'hello.'
101  13:     UTCTime 23/02/2015 01:09:00 GMT
      :     }
      :   }
0 warnings, 0 errors.
```

NB! `dumpasn1` fails to decode negative integers and outputs bitstrings in reverse order.

```
asn1_tag_explicit(asn1_sequence(asn1_set(...)+asn1_boolean(true)+...), 0)
```

Type-Length-Value: Length

- asn1_len(content_str):
  - If the number of content octets <= 127 then length octet encodes number of content octets.
  - Else the most significant bit of the first length octet is set to 1 and other 7 bits describe number of length octets following.
    - The following length octets encode the length of content octets in Big Endian byte order.

Example:

```
Length 126:  01111110
Length 127:  01111111
Length 128:  10000001 10000000
Length 1027: 10000010 00000100 00000011
                      (4 << 8) | 3
                           = 1027
```

ASN.1 DER encoding

- `asn1_boolean(bool)`:
  - Encodes boolean value
  - Universal, primitive, tag 1 (00 0 00001)
  - Content octet contains `0x00` for `FALSE` and `0xff` for `TRUE`
- `asn1_integer(int)`:
  - Encodes integer (encoder must support only positive integers)
  - Universal, primitive, tag 2
  - Big-Endian two's complement integer encoding:
    - If the most significant bit of MSB for a positive integer is 1 then left-pad with zero (`0x00`) byte

```
INTEGER: 140
DER: 00000010  00000010   00000000   10001100
        Type     Length    Padding    Integer
```

ASN.1 DER encoding

- `asn1_bitstring(str_of_bits)`:
  - Encodes an arbitrary string of bits (e.g. '010101')
  - Universal, primitive, tag 3
  - Bitstring is right-padded with zero bits to form octet string
  - First octet of content octets gives the number of padding bits

```
BIT STRING: 010101
DER: 00000011  00000010     00000010     01010100
       Type     Length  Padding-length Padded-bitstring
```

- `asn1_octetstring(byte_str)`:
  - Encodes an arbitrary string of octets
  - Universal, primitive, tag 4
- `asn1_null()`:
  - Denotes a null value
  - Universal, primitive, tag 5
  - No content octets

ASN.1 DER encoding

- `asn1_objectidentifier(list_of_oid_components)`:
  - An object identifier, which is a sequence of integer components
  - Universal, primitive, tag 6
  - First octet has value: `40 * value1 + value2`
  - Following octets encode `value3, ..., valuen`:
    - Each value is encoded using 7 righmost bits
    - Each octet's msb (leftmost bit) except for the last is 1

  Example:

```
OBJECT IDENTIFIER: 1.2.840 (US (ANSII))
0000 0110  0000 0011     0010 1010    1 0000110     0 1001000
 Type       Length        40*1+2           6             72
                                      (6 << 7) | 72 = 840
```

- `asn1_sequence(der_bytestr)`:
  - Encodes ordered collection of one or more types
  - Universal, **constructed**, tag 16
  - Content octets contain DER encoded collection of types

ASN.1 DER encoding

- `asn1_set(der_bytestr)`:
  - Encodes unordered collection of one or more types
  - Universal, **constructed**, tag 17
  - Content octets contain DER encoded collection of types
- `asn1_printablestring(bytestr)`:
  - Encodes an arbitrary string of printable characters `[a-zA-Z0-9' ()+,-./:=?]`
  - Universal, primitive, tag 19
  - Content octets contain printable string octets
- `asn1_utctime(date_str)`:
  - Encodes "coordinated universal time" or Greenwich Mean Time (GMT) in form "YYMMDDhhmmssZ"
  - Universal, primitive, tag 23
  - Content octets contain string representation of time

ASN.1 Tagging

ASN.1 notation may be ambiguous:

```
Ambiguous ::= SEQUENCE {
  val1 INTEGER OPTIONAL,
  val2 INTEGER OPTIONAL
}
```

Unable to decode if encoded structure contains only one value!

Fix (tag the values):

```
unambiguous ::= SEQUENCE {
  val1 [1] IMPLICIT INTEGER OPTIONAL,
  val2 [2] EXPLICIT INTEGER OPTIONAL
}
```

- IMPLICIT overwrites the existing type byte
- EXPLICIT adds type and length (encapsulates original TLV)

http://powerasn.ncottin.net/download/ASN1_Tagging.pdf

ASN.1 DER encoding

- `asn1_tag_explicit()`:
  - Encodes/encapsulates/tags any data type
  - **Context-defined, constructed**, tag *n* (right-most 5 bits)
    - No need to implement support for tag > 30
  - Content octets contain DER-encoded data

```
>>> asn1 = asn1_tag_explicit(asn1_sequence(asn1_null()), 5)
>>> open('asn1', 'w').write(asn1)
```

```
$ dumpasn1 asn1
  0   4: [5] {
  2   2:   SEQUENCE {
  4   0:     NULL
        :     }
        :   }
```

Banned functions

Your solution should **not** use:

- functions: `bin()`, `int()`, `hex()`, `str()`, `bytearray()`, `divmod()`
- exponentiation: `**`
- division and modulus: `/`, `%` (unless needed for computing bitstring padding size)

Use bitwise operations as much as possible!

For example, to convert bitstr to int:

```
i = 0
for bit in '010001':
    i<<=1
    if bit=='1':
        i|= 1
```