

MTAT.07.017
Applied Cryptography

Abstract Syntax Notation One (ASN.1)

University of Tartu

Spring 2019

Abstract Syntax Notation One

“ASN.1 is a standard interface description language for defining data structures that can be serialized and deserialized in a cross-platform way. It is broadly used in telecommunications and computer networking, and especially in cryptography.”

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

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  
}
```

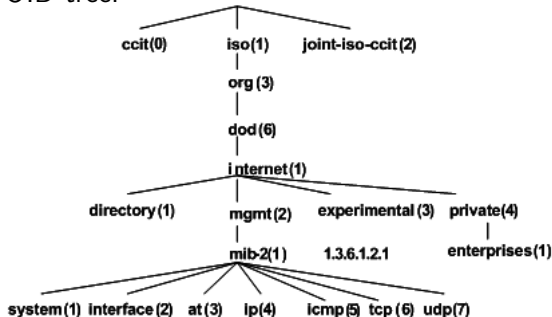
ASN.1 OBJECT IDENTIFIER

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)
- Distinguished Encoding Rules (DER)
- Packed Encoding Rules (PER)
- XML Encoding Rules (XER)
- JSON Encoding Rules (JER)

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'
```



Task: ASN.1 DER encoder – 10p

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):

```
$ 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
: Error: Spurious zero bits in bitstring.
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
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
: }
: }
```

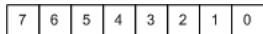
NB! dumpasn1 fails to decode negative integers and outputs bitstrings in reverse order.

```
0 warnings, 1 error.
```

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

```
$ ./asn1_encoder.py asn1.der
```

Type-Length-Value: Type



7	6	class
0	0	UNIVERSAL
0	1	APPLICATION
1	0	context-defined
1	1	PRIVATE

5	form
0	primitive
1	constructed

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

...

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>

Type-Length-Value: Length

- `asn1_len(value_str): - 1.5p`
 - If the number of value octets < 128 then length octet encodes the number of value octets
 - Else the most significant bit of the first length octet is set to 1 and the remaining 7 bits encode the number of length octets that follow
 - The following length octets encode the number of value octets (use `nb()` without the `length` parameter)

Example:

Length 126: 01111110

Length 127: 01111111

Length 128: 10000001 10000000

Length 1027: 10000010 00000100 00000011

(4 \ll 8) | 3

= 1027

ASN.1 DER encoding

- `asn1_boolean(bool)`: – 0p
 - Encodes boolean value
 - Universal, primitive, tag 1 (00 0 00001)
 - Value octet contains 0x00 for FALSE and 0xff for TRUE
- `asn1_integer(int)`: – 1p
 - Encodes integer (only positive integers must be supported)
 - Universal, primitive, tag 2
 - Two's complement integer encoding:
 - Convert integer to bytestring using `nb()` without the `length` parameter
 - Integer value 0 is encoded as zero byte (not empty bytestring)
 - If the most significant bit of MSB for a positive integer is 1 then prepend zero (0x00) byte

INTEGER: 140

DER:	00000010	00000010	00000000	10001100
	Type	Length	Padding	Integer

250,000 Estonian ID cards could be faulty

<https://news.err.ee/116849/250-000-estonian-id-cards-could-be-faulty>



A **coding mistake by the Certification Center**, the company behind the software of ID cards, means 250,000 ID cards could cause problems for users in the future.

The problem concerns Estonian ID cards issued between September 2014 and September 2015, and if not fixed, will mean users are unable to use ID cards with the new version of the Google Chrome browser.

“We let a fault slip through our software development process,” Certification Center head Kalev Pihl told Postimees. The problem surfaced when Google worked out a **new version of Chrome, which has more detailed checks.**

ASN.1 DER encoding

- `asn1_bitstring(str_of_bits)`: - 2p
 - Encodes an arbitrary bitstring value (e.g. '010101')
 - Universal, primitive, tag 3
 - Bitstring is right-padded with zero bits to form full octet string
 - First octet of value octets encodes number of padding bits

BIT STRING: 010101

DER:	00000011	00000010	00000010	01010100
	Type	Length	Padding-length	Padded-bitstring

- `asn1_octetstring(byte_str)`: - 0.2p
 - Encodes an arbitrary string of octets
 - Universal, primitive, tag 4
- `asn1_null()`: - 0.2p
 - Denotes a null value
 - Universal, primitive, tag 5
 - No value octets

ASN.1 DER encoding

- `asn1_objectidentifier(list_of_oid_components)`: – 3p
 - An object identifier, which is a sequence of integer components
 - Universal, primitive, tag 6
 - The first value octet has value: $40 * comp1 + comp2$
 - The following value octets encode `comp3`, `comp4`, ...
 - Each component is encoded using 7 rightmost bits of the octets
 - Each octet's 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
			$(6 \ll 7) 72 = 840$	

- `asn1_sequence(der_bytestr)`: – 0.2p
 - Encodes ordered collection of one or more types
 - Universal, **constructed**, tag 16
 - Value octets contain DER encoded data

ASN.1 DER encoding

- `asn1_set(der_bytestr)`: – 0.2p
 - Encodes unordered collection of one or more types
 - Universal, **constructed**, tag 17
 - Value octets contain DER encoded data
- `asn1_printablestring(bytestr)`: – 0.2p
 - Encodes an arbitrary string of printable characters
[a-zA-Z0-9' ()+, - . / : = ?]
 - Universal, primitive, tag 19
 - Value octets contain printable string octets
- `asn1_utctime(date_str)`: – 0.2p
 - Encodes "coordinated universal time" (GMT – Greenwich Mean Time)
 - Universal, primitive, tag 23
 - Value octets contain string representation of time in form "YYMMDDhhmmssZ"

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 is to tag the values:

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

- IMPLICIT overwrites the existing type byte of TLV
- EXPLICIT prepends type and length bytes (encapsulates original TLV)

ASN.1 DER encoding

- `asn1_tag_explicit(der, tag):`
 - Tags/encapsulates any data type
 - **Context-defined, constructed**, tag n (5 rightmost bits)
 - No need to implement support for tag > 30
 - Value 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: `**`, `pow()`
- division and modulus: `/`, `%` (unless needed for computing bitstring padding size)

Use bitwise operations as much as possible!

For example, to convert `str` containing bit representation to `int`:

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

As a general rule for all homeworks: encoding values to “bin” and “hex” representation is allowed only for printing out unprintable byte values.