A-1 In the lecture notes, the type Course has been imported from the LTH-module. Write a suitable definition of this type. You must use at least three different types in your definition.

```
Course ::= SEQUENCE {
              courseName VisibleString,
              courseCode VisibleString,
              HP REAL,
              professors SEQUENCE OF SEQUENCE {
                          profName UTF8String (SIZE(1..40)),
                          profEmail UTF8String (SIZE(4..70))
             }
}
```

A-8 For each of BER, CER, DER, PER, and XER, give the full name and summarize the main idea behind that particular encoding rule.

BER - Basic Encoding Rules: The original encoding rules for ASN.1. The main characteristic of the BER encoding is that it is simple but it is quite inefficient. BER uses a Type-Length-Value approach in the encoding. Quite inefficient because data that fit into single bits are expanded to a minimum of 8 bits. The BER encoding rules has in many cases several different possible encodings for the same thing

CER - Canonical Encoding Rules: Is a restricted variant of BER for producing unique encoding for the elements. Length encoding is indefinite for all constructed encodings. Shortest possible definite form for primitive encodings. Strings are always split into chunks of 1000 bytes.

DER - Distinguished Encoding Rules: Is a restricted variant of BER for producing unique encoding for the elements. Length always encoded as shortest possible definite form. Strings are never split, they are encoded in primitive form.

PER - Packed Encoding Rules: Because BER, CER and DER are not very efficient because they use more bits then are actually needed to store the data, PER was introduced. Does not explicitly state the type because it is stated in the ASN.1 structure. Restricted integers could be encoded with few bits etc. Used for applications where memory is limited.

XER - XML Encoding Rules: Makes it possible to define values using XML markup in which the element tags are derived from the ASN.1 type names and identifiers present in an ASN.1 specification.

A-9 For the long definite form, what appears to be the maximum length possible to encode?

The number of bytes that gives the length is given by the first byte in the form `1XXXXXXX`. This means that we can have a length specified by 127 bytes. Having all bits in these bytes set to 1 would give us $8 * 127 = 1016$ bits. The maximum value is therefor $2^{1016} - 1$.

A-11 Decode the following: 3A 82 00 10 1A 01 73 1A 02 65 63 1A 81 06 75 72 69 74 79 21. What encoding rule is it?

This results in the string **security!** according to BER.

According to the lecture slides the decimal value for `VisibleString` is 26, this equates to `1A` in hex. Looking at the string we have three different `1A` entries which indicates three different strings. The first part before the first string entry is `3A 82 00 10`. Following the TLV scheme the type should be one byte which then is `3A`. The bit representation of this is `0011 1010`. The third bit from the left is indicating that this should be a constructed P/C, e.g. a sequence. The rest is the actual tag which then gives `1 1010` which equates to `1A`, i.e. a sequence of VisibleStrings. The first entry is listed as `1A 01 73` which equates to a `VisibleString` of length 1 with the hex value of 73 which equates to ascii **s**. The second entry is listed as `1A 02 65 63`(using the same logic as with the first entry) equates to ascii characters **ec**. The final entry is listed as `1A 81 06 75 72 69 74 79 21`. I did not figure the 81 out but after that comes 06 for 6 bytes which is the last `75 72 69 74 79 21` which equates to **urity!** so the final string then becomes **security!**.

A-19 Consider the SignedData type in CMS, which is used to sign data. Where is the actual signed data (e.g., a document, a letter or a contract) given?

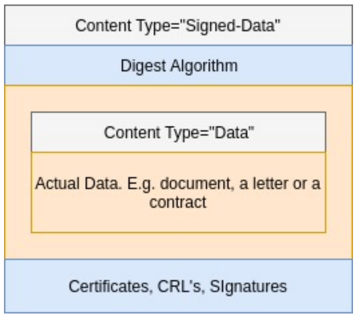
Content Type="Signed-Data"
Digest Algorithm
Content Type="Data"
Actual Data. E.g. document, a letter or a contract
Certificates, CRL's, SIgnatures

Figure 1: Location of the actual data in a SignedData type

I have tried to visualize it in figure 1. This would equate to the data in the `encapContentInfo` field of the SignedData type. The `encapContentInfo` is a sequence of `ContentType` and a optional `OCTET STRING`. See slides page 29.

A-20 Consider the SignedData type in CMS. The digestAlgorithms are given as a "SET OF DigestAlgorithmIdentifier". Since a "SET OF" does not have a particular order, how can we know which digest algorithm corresponds to which signer? Or do we not care?

The set of DigestAlgorithmIdentifier is the union of all algorithms used by all signers. We do not care about the order because The the algorithm used by each signer is given in the set of SignerInfo which maps the signer identifier to the digest algorithm it used.

A-23 In PKCS #12, assume that we want to represent a private key. It should be privacy and integrity protected using a password. What is the minimum number of ContentInfo types we have to define in order to produce a valid PFX? Which are the ContentTypes we should use?

So the PFX is defined by:

```
PFX ::= SEQUENCE {
            version INTEGER,
            authSafe ContentInfo,
            macData MacData OPTIONAL
}
```

The MacData is used when Password Intergrity Mode, which we require, but it itself does not have any ContentInfo type that the question is asking for so I will disregard it.

So authsafe, the ContentInfo is what is interesting. Since we use Password integrity Mode we the ContentInfo is Data and we should use the type `AuthenticatedSafe ::= SEQUENCE OF ContentInfo`.
Since we use password privacy mode the ContentInfo should be `EncryptedData`. The `EncryptedData` type has content, this should be `SafeContents` because that we use password privacy mode.

`SafeContents ::= SEQUENCE OF SafeBag` where `SafeBag` holds the actual information. Since we have no requirement to use the shrouded variant the `KeyBag` should be good enough.

I have tried to illustrate this in figure 2 on the next page and I count the number of ContentInfo types to be the authSafe in the `PFX`, which itself is a sequence of ContentInfos. We Should only have one type in the sequence so this adds up to two. I am unsure of which of these types count as actual ContentInfo.

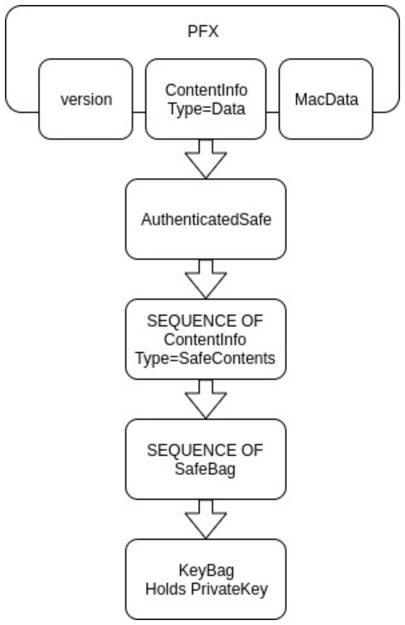

Figure 2: Structure when representing a private key

A-27 Can a DoS attack stop a CRL update from reaching a potential victim? How should that victim behave when the nextUpdate time has been reached and no update has arrived?

Yes a DoS attack can stop a CRL update from reaching a potential victim. This leads to that the victim has an old version of the CRL and adversaries can then use revoked certificated to do malicious stuff because the victim is not aware. This is a reason behind the nextUpdate field of the CRL. If the current time passes the nextUpdate the victim should invalidate certificates that are checked against the expired CRL to guarantee that no adversary can do something that harms the victim.