Structureel testen van Quick DER

> *Met CMake, CTest en CDash wordt het wel heel simpel om Quick DER te testen. Kunnen we een hele stapel tests genereren op basis van de ASN.1 syntax?*

Wat we moeten doen is allerlei varianten van elke syntax genereren, parsen en reconstrueren, en zien dat er hetzelfde uit komt.

Genereren van de binaries zou niet met Quick DER moeten worden gedaan (helaas). Mogelijk kan het wel op meta-niveau worden gedaan, en kan zo een lijst met `add_test()` statements worden geconstrueerd. (Die zitten dan wel op een rare plek in de interactie...)

Eigenlijk lijkt het een betere test, en ook vermijdt het veel extra code, om niet van DER uit te gaan maar van de vorm in een applicatie; en om dan de omzetting te doen van applicatie naar DER naar applicatie, dus `der_pack()` gevolgd door `der_unpack()`.

De zo gegenereerde testprogramma's kunnen de varianten opsommen en aangeroepen worden met specifieke tests. Het lijkt zinvol om een test-wrapper te maken per vertaalde ASN.1 module, zodat klachten lokaal aan een module blijven en ook een explosie aan opties vermeden wordt. Dat betekent dat elke module ook in staat moet zijn om een representatieve binary aan te dragen, eentje die goede kans heeft erdoor te komen. Dit kan dan wel weer in een `#define` worden gestopt.

Het guarden van ruimte rondom de `der_pack()` is ook een slim idee dat in elke test thuishoort. Idem, de ruimte rond een `dercursor` array kan ook met guards worden getest.

De nesting van een ASN.1 definitie regelt eigenlijk automatisch het testen ervan. Ook hier geldt weer dat het zinvol kan zijn om per symbool te testen, zodat minder een beroep wordt gedaan op de explosie aan test cases (die redelijkerwijs orthogonaal verondersteld mogen worden).

Test programma's kunnen de hele module testen of, gegeven een argument, specifiek een bepaalde definitie of, gegeven een testnummer `13/24` kan een heel bepaalde test worden gedraaid, in dit geval nummer `13` uit `24` (we stellen `0/24` en `24/24` aan elkaar gelijk). Het vermelden van `24` is zinvol omdat gekeken kan worden of er inderdaad zoveel tests zijn; zo niet, dan volgt daar een foutmelding op en kan de test-infra worden aangepast.

Structuren

Deze zijn de hoofdinspanning van Quick DER, dus heel veel zinvoller om te testen dan al het andere.

- `SEQUENCE` en `SET`, gewoon binnengaan
- `SEQUENCE OF` en `SET OF`, test vooral ook de lege gevallen; zelfs als de semantiek van een applicaties ze verbiedt zijn het nog dingen die in DER goed moeten gaan
- `CHOICE`, alle varianten afzonderlijk testen
- `OPTION`, probeer de gevallen met en zonder
- `ANY`, test een aantal soorten die spannend lijken, inclusief lege gevallen

Elementaire waarden

- `INTEGER`, genereer gekke waarden zoals `0` in 0 bytes, een negatieve waarden `0x80`, `0xc0`, `0x8000000` en `0xc0000000`, en ook positieve zoals `0x40`, `0x00` en `0x40000000` (die in topbits verschillen dus).
- `BIT STRING`, test de varianten met bits "over", eventueel ook de gevallen die (onterecht) waardes 1 hebben
- diverse textuele `STRING` formaten, genereer lege strings en waar mogelijk UTF-8 gevallen.