Annotated Schema Example – DNMP

DNMP (Distributed Network Measurement Protocol) is a protocol for securely accessing an NDN network's operational and performance data. It was designed to provide fine-grained access control yet be robust and easy to configure. This is the trust schema developed for DNMP.

The VerSec language allows the schema to be described in any order. This schema starts with defining the top-level 'namespace' used for network operational communications. This string will be the first component of every publication name in this namespace. It is also the name of the self-signed certificate used as the trust anchor of those publications' signing chains. This name should be unique to the network it's used on but there's no reason to make it either globally unique or routable.

If you're satisfied with the default roles and policy, all that's required to bring up VerSec and DNMP on a new network is to pick this name then run a setup script that generates the initial signing certs.

This schema is for Alice's home IoT net, hence the name:

```
// this network's NetOps namespace
_network: "AliceNetOps"

// the trust anchor cert name
netCert:  _network/_keyinfo
```

DMNP may be one of many netOps services so it uses the sub-namespace `dnmp` to avoid conflicts with others. The signing cert `dnmpCert` represents the top-level namespace delegating control over publishing in the dnmp sub-namespace to the holder of `dnmpCert`:

```
// subdomain associated with this schema & associated cert
_domain: "dnmp"
dnmpCert: _network/_domain/_keyinfo <= netCert
```

Principals in the DNMP model are *people* (who issue *commands* via CLI or bespoke programs) and *devices* (who *reply* to commands via an on-device *NOD* (Network Observer Daemon) agent). The `dnmpCert` is used to issue signing certs that assign *roles* to the principals that can issue commands. The `roleCert` definition below specifies the components and signer of any kind of role cert. The following two lines define two specific roles: *operator* and *user*.

```
// Certificates and signing chain for roles that can issue commands
roleCert: _network/_domain/_role/_roleId/_keyinfo <= dnmpCert
opCert:   roleCert & { _role: "operator" }
userCert: roleCert & { _role: "user" }
```

The `#command` definition below specifies the component names of a command publication then binds some of those components to particular values: *_topic* will always be the string "command", *_origin* will be filled with a string identifying the pid and host of the process that published the command and *_cTS* will be set to a 64-bit timestamp specifying when the command was published (used for replay-prevention among other things).

The next two lines define the types of commands that operators and users can send: Operators can send any command to any or all NODs while users can send any command to the NOD on the same machine (`target:"local"`) or a *ping* command to anything (`pType:"Pinger"`).

Note that the `ocommand` definition shows that VerSec supports not just Role-based Access Control but full Attribute-based Access Control conforming to NIST SP 800-162.

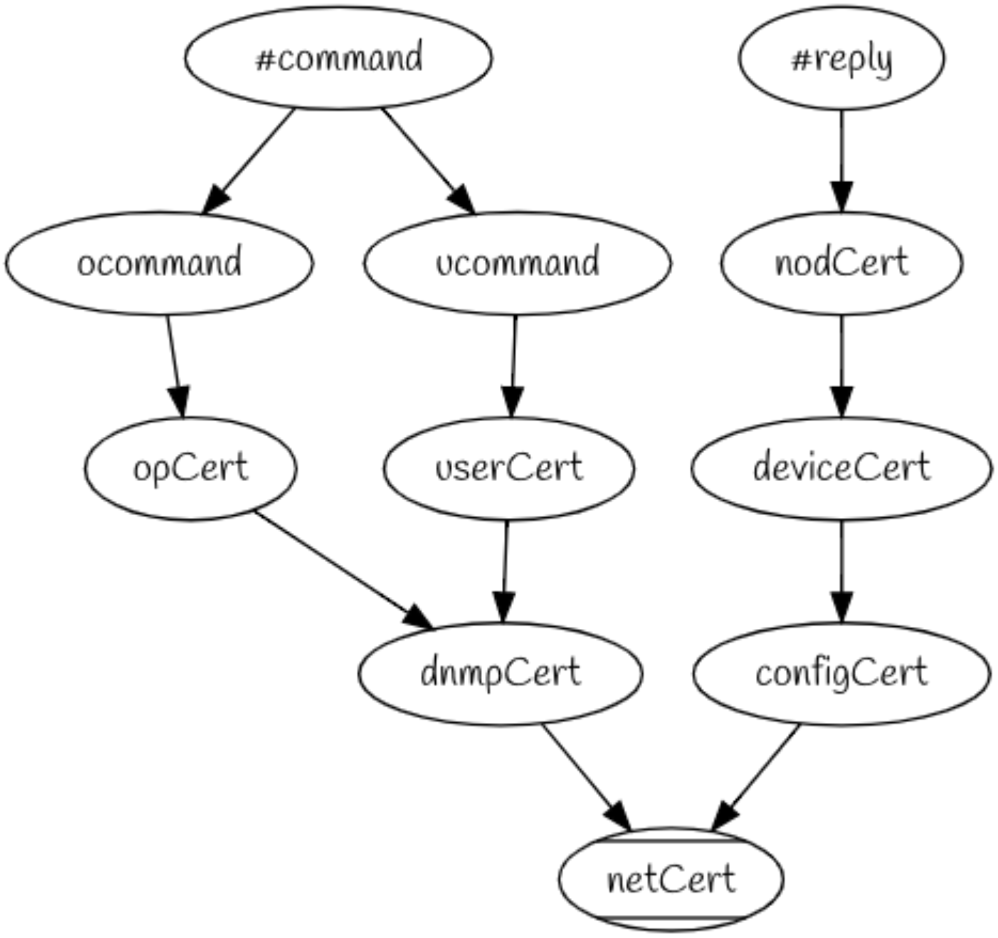
```
// command publication definition
#command: _network/_domain/target/_topic/_roleId/pType/pArgs/_origin/_cTS &
                { _topic: "command", _origin: sysId(), _cTS: timestamp() }

// Operators can issue any command
ocommand: #command <= opCert

// Users can issue any command to local nod and ping commands to any nod
ucommand: #command & { target: "local" } | { pType: "Pinger" } <= userCert
```

Most local networks today use multicast-capable media like WiFi or Ethernet. DNMP takes full advantage of this by *not* using a point-to-point, conversational transport like TCP but rather a broadcast-friendly, pub-sub transport based on set-reconciliation. Using multicast both simplifies the communication and makes it more robust (since communication no longer depends on establishing heavy-weight abstractions like IP endpoint addresses or routing spanning trees). But, in a conversation, questions are matched with answers implicitly – one endpoint asks, the other answers and, with only two parties, there's no ambiguity. This doesn't work for non-conversational broadcast so DNMP replies 'repeat the question' but change its `topic` from `"command"` to `"reply"`. Thus the command publisher knows the prefix that will start all replies to each command and can subscribe to it just before issuing the command. The `replace()` operator in the `#reply` definition below implements the NOD side of this convention.

Also, note that communication between people and devices is asymmetric – people can issue commands but can't reply to them while devices can do nothing but reply. This asymmetry limits the attack surface and was deliberate.

```
// reply publication definition
#reply: replace(#command, _topic, "reply")/_nodId/_rTS & { _rTS: timestamp() } <= nodCert
```

Devices are configured and managed at the network level, not the DNMP level, so they have a different signing chain than people. Devices are configured and given their identity (`deviceCert`) by a person authorized to configure and enroll devices (`configCert`) by the network (`netCert`). This chain is described below. The overall signing graph looks like:



```
// certificates and signing chains for roles that reply to commands
nodCert:    _network/_domain/"nod"/_nodId/_keyinfo <= deviceCert
deviceCert: _network/"device"/_devId/_keyinfo <= configCert
configCert: _network/"config"/_confId/_keyinfo <= netCert
```

This final publication reflects the needs of the low-level Pub-Sub transport `syncps`. It uses NDN Interest/Data packets to get publications from publishers to subscribers. That means it needs to know how to map the DNMP-level notion of a publisher's targeted subscribers, `target`, to an NDN prefix that will reach all those subscribers. The `#wirePrefix` definitions are that map. They use an NDN prefix of `/localhost/dnmp/`*target* if the targets are restricted to the local machine and `/localnet/dnmp/`*target* otherwise (which reaches the entire network).

```
// Prefix used at the NDN Interest/Data level to sync this collection.
#wirePrefix: _ndnprefix/_domain/target
wpLocal: #wirePrefix & { _ndnprefix: "localhost", target: "local" }
wpOther: #wirePrefix & { _ndnprefix: "localnet" }
```