Trust schemas and ICN: key to secure Home IoT

Kathleen Nichols
September 24, 2021
ACM Information-Centric Networking ‘21

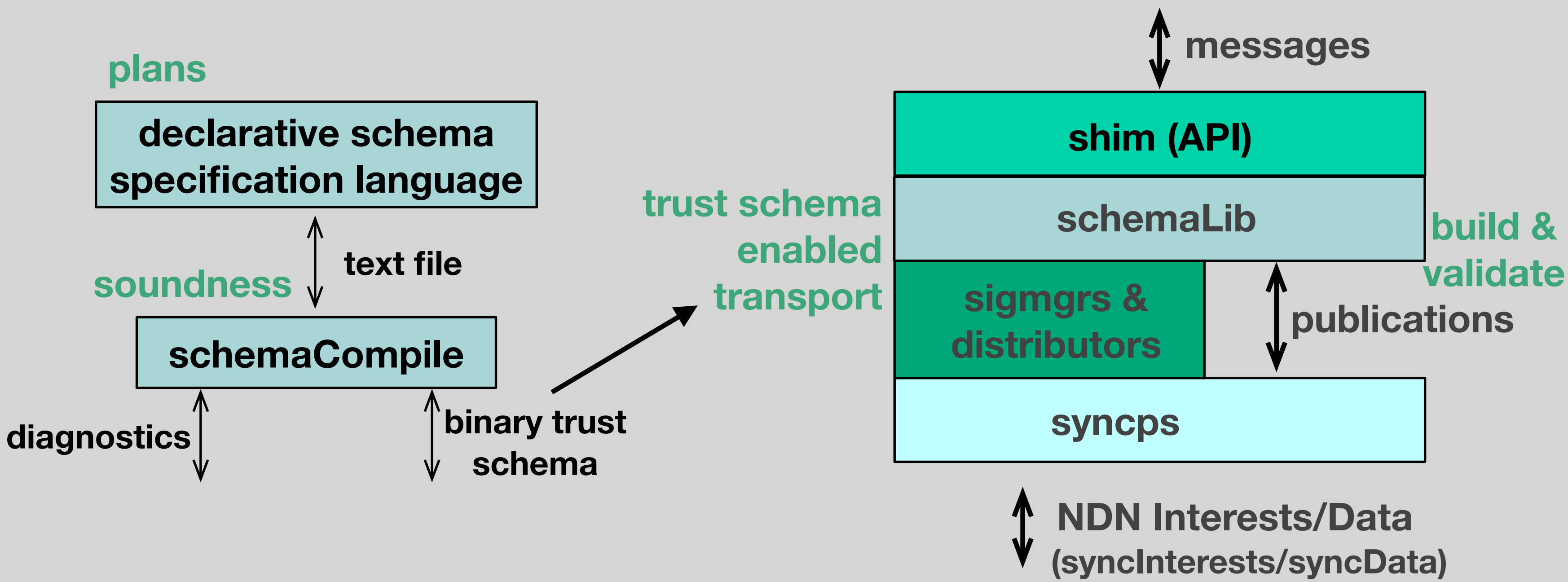
Trust schemas express rules for **who** gets to say **what** to **which**

- Identities are conferred through signing certificates with complete signing chains to give the **who**
- Strictly defined Data name structures linked to components of an identity's signing chain constrain **what** can be communicated
- Rules that link valid Data format to legal signers let recipients decide **which** communications to accept

For a particular *domain* where the rules and certificates apply

Employ trust schemas like building plans that can be used as specifications to check for code compliance, to build, and for an inspector to validate compliance

This requires tools and methodology



Evolving tools, documentation, examples at https://github.com/pollere/DCT

home IoT makes an attractive use case

- well-specified communications for entities
- single administrative authority (single root of trust)

e.g., SmartThings *capabilities* represent the functions of devices, with *commands* to control device functions and *attributes* that represent the state or properties of a device (smartthings.com)

commands and attributes define **what** a home IoT network will communicate and are defined in JSON schema, for example:

```
"commands": {
    "setColorTemperature": {
        "arguments": [
            {
                "name": "temperature",
                "optional": false,
                "schema": {
                    "type": "integer",
                    "minimum": 1,
                    "maximum": 30000
                }}]}}
```

```
"attributes": {
    "colorTemperature": {
        "schema": {
            "type": "object",
            "properties": {
                "value": {
                    "type": "integer",
                    "minimum": 1,
                    "maximum": 30000 },},
            "additionalProperties": false
        },
        "required": [
            “value" ] } }
```

application communications determine publication names

netID/**capability/topic/loc/args/**origin/msgID/sCnt/mts

device functions e.g., switch, light, lock

any additional information required for this topic

message type e.g., command, attribute, event

distinguishes which capability and topic, can indicate a room or floor of a building, a particular device or class of devices, a network, or local to keep publications off network

non-bold components specified by DCT/mbps conventions (see paper, GitHub)

- components that are completely determined by the trust schema from available certificates derive content from a component in a cert in the signing chain, e.g., *capability*. These have a leading “_”.
- components can specify a function that the builder calls at build-time, e.g., *origin* can be filled with sysId() call. These have a leading “_”.
- components that are supplied (from app through shim) at the time the pub is built, e.g., *topic*, do not have a leading “_”

write *dctIoT.trust* rules in terms of pubs and identities

```
_net:    "houseNet"
// Publication definition
#Report: _net/_cap/topic/_devId/args/_origin/mID/sCnt/mts & {_origin: sysId()}
switchCert: capabilityCert & {_cap: "switch"}
lightCert: capabilityCert & {_cap: "light"}
lsState: #Report & { topic: "attribute", args: "on"|"off" } <= switchCert | lightCert
lightEvent: #Report & { topic: "event", args: "on2off"|"off2on" } <= lightCert
// For a paired by _devTag (e.g., "sink") switch and light
#tagCommand: _net/cap/topic/_devTag/args/_origin/mID/sCnt/mts & {
    topic: "command", _origin: sysId() }
lightTagCmd: #tagCommand & { cap: "light", args: "on"|"off" } <= switchCert
// A programatic command can go to any 'loc'
#prgCommand: _net/cap/topic/loc/args/_origin/mID/sCnt/mts & {
    topic: "command", _origin: sysId() }
lightOwnCmd: #prgCommand & { cap: “light”, args: "on"|"off"|"report"} <= ownerCert
capabilityCert: _net/_devTag/_cap/_keyinfo <= deviceCert
deviceCert: _net/_devType/_devId/_keyinfo <= configCert
ownerCert:     roleCert & { _role: "owner" }
roleCert:    _net/_role/_roleId/_keyinfo <= netCert
configCert: _net/"config"/confId/_keyinfo <= netCert
netCert: _net/_keyinfo
// publication prefix and validator type
#pubPrefix:    _net
#pubValidator: “EdDSA"
// Prefix used for syncData (NDN Interest/Data)
#wirePrefix: _ndnprefix/_net & { _ndnprefix: "localnet" }
#wireValidator: "AEAD"
// components are KEY, keyID, issuerID, and version
_keyinfo: “KEY”/_/"dct"/_
```

schemaCompile output for dctIoT.trust

Parameter values required to build this publication

Tag names used to reference components of pub. E.g., “if (pub["args"] == "on") …

```
Publication #Report:
  parameters: topic args mID sCnt mts
  tags: /_net/_cap/topic/_devId/args/_origin/mID/sCnt/mts
  signing chains:
    chain 0: lsState <= switchCert <= deviceCert <= configCert <= netCert
             lsState[1]==switchCert[2]  lsState[3]==deviceCert[2]
    chain 1: lsState <= lightCert <= deviceCert <= configCert <= netCert
             lsState[1]==lightCert[2]  lsState[3]==deviceCert[2]
    chain 2: lightEvent <= lightCert <= deviceCert <= configCert <= netCert
             lightEvent[1]==lightCert[2]  lightEvent[3]==deviceCert[2]
  templates:
    /"houseNet"/_cap/"attribute"/_devId/args/sysId()/mID/sCnt/mts { switchCert lightCert }
     [ args: on | off ]
    /"houseNet"/_cap/"event"/_devId/args/sysId()/mID/sCnt/mts { lightCert }
     [ args: on2off | off2on ]
```

Signing chains required for each pub use case

Leading “_” in name means must have the same value in pub & certs

This template can be signed by chain 0 or 1

Pub builder puts origin’s system Id here

Legal “args” values for this template item

NDN cert conventions require these components but the schema doesn’t supply tag names to reference them

```
...
Certificate templates:
  cert ownerCert: /"houseNet"/"owner"/_roleId/"KEY"/_/"dct"/_
  cert lightCert: /"houseNet"/_devTag/"light"/"KEY"/_/"dct"/_
  cert switchCert: /"houseNet"/_devTag/"switch"/"KEY"/_/"dct"/_
  cert deviceCert: /"houseNet"/_devType/_devId/"KEY"/_/"dct"/_
  cert configCert: /"houseNet"/"config"/confId/"KEY"/_/"dct"/_
  cert netCert: /"houseNet"/"KEY"/_/"dct"/_

40 strings, 258 bytes (4 overlaps, 247 bytes in stab)
binary schema is 601 bytes
```

next, use trust schema to create a *trust zone*

- the trust schema specifies the format of communications and signing certificates with signing chains that terminate in the same trust anchor
- entities that use the same trust schema and have associated legal signing certificates form a trust zone that remains secure without the need for physical network isolation
- DCT provides some utilities to help with this:

make a local trust anchor:	`make_cert -s EdDSA -o house.root houseNet`
make a binary trust schema:	`schemaCompile -o dcIoT.scm dctIoT.trust`
make a trust schema cert:	`schema_cert -o dctIoT.schema dctIoT.scm house.root`
make a role cert:	`make_cert -s EdDSA -o alice.cert houseNet/owner/alice house.root`
make an identity bundle:	`make_bundle -o alice.bundle house.root dctIoT.schema +alice.cert`

- The *alice* bundle has three certificates and includes her private signing key

start *app* from command line: `%app alice.bundle`

- Now alice can join the trust zone!

alice's app connects to the trust zone

- app pubs will handled through a syncps collection houseNet/pubs
- but first a related collection, houseNet/certs, is used by a signing certificate *distributor* to make sure the other apps have all the certs in this app's signing chain and it has theirs
- if the schema requires content privacy (AEAD symmetric encryption), collection houseNet/key is used by a group key distributor to elect a key maker then distribute the ephemeral shared secret key to all holders of a valid signing cert (new entities are added as their signing cert is published)

signing cert distributor

1. joins houseNet/certs
2. publishes alice's signing cert chain
3. validates any received signing chains using trust schema and trust anchor (on-going)

on successful publication

group key distributor

1. joins houseNet/key
2. **if** no data there, creates a group key, encrypts with public key of each signing cert received, and publishes the list
   else find my copy of group key in the data and decrypt

have group key

indicate to app: connected to zone

# of note

- **alice** has no a priori knowledge of other entities in the the trust zone nor they of her
- application has no interaction with the certificate and key distribution; these are invoked by *schemaLib* methods based on the *sigmgrs* selected
- if AEAD is specified, a syncData packet in the pub collection is discarded if it isn’t AEAD signed with a known key
- if EdDSA is specified
  - for “wire”, a syncData packet in the pub is discarded if it isn’t EdDSA signed by a known signer
  - for pub, the publication is discarded if it isn’t EdDSA signed by a known signer (cryptographic validation) or if the signer’s chain isn’t consistent with the publication’s name according to the schema (“structural” validation)

Further work

– more expressive versec/schemaCompile (struct components)
– more efficient broadcast communications “on wire”
– modules to distribute trust zone geographically
– “hierarchical” and connected trust zones

Opportunities

– easy to add sigmgrs for security research
– improved group key distributor
– provisioning and on-line updates for trust schemas, signing certs
– shims for other communications models
– tools to go from rules to versec
– tools to go from application communications to versec