### The "Client/Server" Set of Drafts

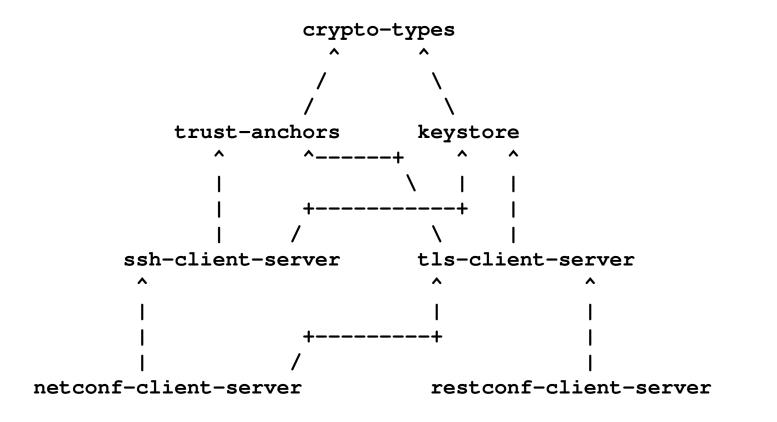
draft-ietf-netconf-crypto-types-00 draft-ietf-netconf-trust-anchors-00 draft-ietf-netconf-keystore-05 draft-ietf-netconf-ssh-client-server-06 draft-ietf-netconf-tls-client-server-06 draft-ietf-netconf-netconf-client-server-06

#### NETCONF WG IETF 102 (Montreal)

### Since IETF 101

- adopted the "crypto-types" and "trust-anchors" drafts
- did not unadopt the "keystore" draft
- all drafts updated and submitted as a set
- open issues linger...

### Relationship between Drafts



#### Keep trust-anchors separate from keystore?

#### History:

- Originally, ietf-keystore had both keys and trust-anchors together.
- ietf-trust-anchors was created in an attempt to get rid keystore
- But we wound up keeping keystore, and trust-anchors remain separate.

#### Tradeoffs:

- Keeping separate provides more applicable names (keystore just stores keys) and provides some modularity (drafts can update independently).
- Bringing together would be one less module do deal with...

- 1. Keep modules separate (current drafts)
- 2. Bring together again (like the old keystore draft)

#### Keep "local-or-keystore" keys?

The "local-or keystore" construct came from wanting applicationspecific keys (keys that are not shared for any other purpose).

This choice statement accurately configures single-use keys, but it makes for rather busy models.

An alternative could be to instead have all keys in keystore, and let the application/operator deal with ensuring that some keys are not referenced more than once.

- 1. Keep "local-or-keystore" (keystore MAY be implemented)
- 2. Eliminate the "local" option (keystore MUST be implemented)

# Assuming "local-and-keystore", how to disable support for the "local" choice?

- 1. add "if-defined 'not keystore-implemented'" to the "local" choice. (a \*global\* on/off switch, not per use of the grouping)
- 2. add "if-defined 'local-keys-supported'" to the "local" choice. (a \*global\* on/off switch, not per use of the grouping)
- 3. do nothing to the grouping definition, let downstream modules augment-in their own if-feature statements. (a peruse switch, but what would the ssh/tls-client-server drafts do when using the grouping?!)
- 4. Don't attempt to disable the "local" choice. (effectively same a option 3)

# Should some of Keystore's groupings be moved to crypto-types?

The following groupings *are not* Keystore-specific:

- public-key-grouping
- asymmetric-key-pair-grouping
- asymmetric-key-pair-with-certs-grouping \*
- end-entity-cert-grouping \*
- trust-anchor-cert-grouping \*?

The following groupings *are* Keystore-specific:

- local-or-keystore-asymmetric-key-grouping
- local-or-keystore-asymmetric-key-with-certs-grouping \*
- local-or-keystore-end-entity-certificate-grouping \*
- \* Note that the groupings with an asterisk define a notification ("certificate-expiration")

- 1. move non-Keystore-specific groupings to crypto-types
- 2. keep non-Keystore-specific groupings in Keystore module

# Should algorithm identities be moved from ietf-[ssh/tls]-common to crypto-types?

Uses for identities:

- 1. to define the algorithm used by a key definition, whether configured locally or in the Keystore.
- 2. to constrain the allowed key algorithm types so as to conform to some security policy.
- 3. to specify preference for certain key types by the order in which the types are configured, in case "keys to use"/"keys to check" are unordered lists

Problems:

- three sets of similar identities
- no constraints that the identities used during negotiation have to match the identities for keys that exist.

No solution options yet – any opinions?

## Add a "periodic" feature enabling the initiating peer to optionally support periodic connections?

Currently, the NC/RC client/server modules enable both persistent and periodic connections to be configured, but maybe "periodic" should be optional to implement?

Tradeoffs:

- It seems that periodic connections are not commonly implemented. A feature would primarily be to accommodate that market trend.
- Periodic connections are incredibly useful and, by not having a feature, we might nudge the industry into supporting them more.

- add "periodic" feature (MAY be implemented)
- do not add "periodic" feature (MUST be implemented)

### Add support for TCP Keepalives?

The NC/RC client/server modules currently support configuring SSH/TLS-level keepalives.

But TLS-level keepalives aren't well supported, and so there is a desire to alternately configure TCP-level keepalives...

A budding IETF statement says that, when using a crypto-transport, the aliveness checks SHOULD NOT occur via the underlying cleartext protocol layer.

- 1. do nothing (only crypto-level keepalives can be configured)
- 2. add a flag indicating that the keepalives should be TCP-level
- 3. add a choice statement for the two keepalive options

