Status and Issues for the “Client-Server” Suite of Drafts

draft-ietf-netconf-crypto-types-12
draft-ietf-netconf-trust-anchors-07
draft-ietf-netconf-keystore-14
draft-ietf-netconf-tcp-client-server-03
draft-ietf-netconf-ssh-client-server-16
draft-ietf-netconf-tls-client-server-16
draft-ietf-netconf-http-client-server-00
draft-ietf-netconf-netconf-client-server-16
draft-ietf-netconf-restconf-client-server-16

NETCONF WG
IETF 106 (Singapore)

Since IETF 105

All drafts updated and submitted as a set multiple times!

High-level Updates:

trying to Last Call ASAP

crypto-Types:
- introduction of "key-format" nodes
- removed all non-essential algorithms
- moved remaining algorithms to 3 iana-* modules
- Added a 'config false' "algorithms-supported" list to each of the 3 iana-* modules.

trust-anchors:
- added PSKs and raw public keys (PSKs now removed)"

keystore:
- Added PSK and raw-public-key support
- Made the two "generate-*-key" RPCs be "action" statements instead.

tcp-client-server:
- Moved the common model section to be before the client and server specific sections.
- Added sections "Model Scope" and "Usage Guidelines or Configuring TCP Keep-Alives" to the Common Model section.

ssh-client-server:
- tweaks to if-feature and description statements.
- Updated "server-authentication" and "client-authentication" nodes from being a leaf of type "ts:host-keys-ref" or "ts:certificates-ref" to a container that uses "ts:local-or-truststore-host-keys-grouping" or "ts:local-or-truststore-certs-grouping".

tls-client-server:
- tweaks to if-feature and description statements.
- Updated "server-authentication" and "client-authentication" nodes from being a leaf of type "ts:host-keys-ref" or "ts:certificates-ref" to a container that uses "ts:local-or-truststore-host-keys-grouping" or "ts:local-or-truststore-certs-grouping".

http-client-server:
- in ietf-http-client, removed "protocol-version" and all but the "basic" authentication scheme.
- in ietf-http-server under /client-authentication, added an ability to configure authentication credentials for the "basic" authentication scheme.

netconf-client-server:
- Added refinement to make "cert-to-name/fingerprint" be mandatory false.
- Commented out refinement to "tls-server-grouping/client-authentication" until a better "must" expression is defined.
- Refactored both the client and server modules similar to how the ietf-restconf-server module was refactored in -13 presented in Montreal.

restconf-client-server:
- Added refinement to make "cert-to-name/fingerprint" be mandatory false.
- Commented out refinement to "tls-server-grouping/client-authentication" until a better "must" expression is defined.
- Refactored both the client and server modules similar to how the ietf-restconf-server module was refactored in -13 presented in Montreal.

This Presentation's Focus

1. Key formats
2. Algorithms
3. Raw public keys and pre-shared keys
4. Client authentication: required-or-optional
5. Client authentication: local-or-external
6. Cert-to-name fingerprints

Begin Discussion #1

Key-Formats

New Key-Format Identities

In ietf-crypto-types :

- key-format
  - public-key
    - ssh-public-key
    - subject-public-key-info
  - private-key-format
    - rsa-private-key
    - ec-private-key
    - one-asymmetric-key
    - encrypted-private-key
  - symmetric-key
    - octet-string-key
    - one-symmetric-key
    - encrypted-symmetric-key

one-asymmetric-key, encrypted-private-key } `if-feature "one-asymmetric-key"`

one-symmetric-key, encrypted-symmetric-key } `if-feature "one-symmetric-key"`

Indicates a key's format/structure/encoding
- NOT its algorithm (at least, not intentionally)

Enables key types (e.g., symmetric key) to be expressed in different ways (OctetString vs. OneSymmetricKey).

It MAY be able to support encoding variations (DER vs PEM, and CMS vs. multi-part PEM).

Updated "Key" Groupings

```
grouping public-key-grouping {
  leaf algorithm {
    type iasa:asymmetric-algorithm-type;
    ...
  }
  leaf public-key-format
    type identityref {
      base public-key-format;
    }
    ...
  }
  leaf public-key { ... }
}


grouping asymmetric-key-pair-grouping {
  uses public-key-grouping;
  leaf private-key-format
    type identityref {
      base private-key-format;
    }
    ...
  }
  choice private-key-type {
    leaf private-key { ... }
    leaf hidden-private-key{ ... }
    }
  }
}
```

```
grouping symmetric-key-grouping {
  leaf algorithm {
    type isa:symmetric-algorithm-type;
    ...
  }
  leaf key-format
    type identityref {
      base symmetric-key-format;
    }
    ...
  }
  choice key-type {
    leaf key { ... }
    leaf hidden-key{ ... }
    }
  }
}
```

Any thoughts or concerns?

Begin Discussion #2

Algorithms

Defining a Dictionary of Algorithms

Moved typedefs from ietf-crypto-types to algorithm-specific modules:

- i.e., iana-asymmetric-algs, iana-hash-algs, and iana-symmetric-algs
- easier to understand and maintain.

And each module defines a typedef:

Only "symmetric" is shown, but "asymmetric" and "hash" follow the same pattern.

```
typedef symmetric-algorithm-type {
  type enumeration {
    enum aes-128-cbc { ... }
  }
  type enumeration {
    enum aes-192-cbc { ... }
  }
  ...
}
```

Any thoughts or concerns?

Are "IANA Templates" Possible?

Goal is that these "iana" modules could be "templatized"

- i.e., automatically maintained by IANA

The idea sounds good, but...

- a quick scan of each list shows a multiplicity of RFCs.
  - how would IANA know when an algorithm from a new RFC should be added?

Determining which Algorithms a Server Supports

Each module also defines a config false
list of algorithms supported by the server:

Only "symmetric" is shown, but "asymmetric" and "hash" follow the same pattern.

```
container supported-symmetric-algorithms {
  config false;
  list supported-symmetric-algorithm {
    key algorithm;
    leaf algorithm {
      type symmetric-algorithm-type;
    }
  }
}
```

$\longleftarrow$ typedef from previous slide

1) instead of global lists, have SSH/TLS specific lists?

- i.e.: defined in ietf-[ssh/tls]-common (i.e., "common" would need to be "implemented")
- a total of six lists (i.e., 3 algs x 2 protocols)

2) how are lists to be used?

- cannot use a "leafref" (both due to "config false" as well as being polymorphic)
- have the "algorithm" description statement (in crypto-types) say something like:

"It is RECOMMENDED that each protocol (e.g., SSH, TLS, etc.) makes available a list of the subset of algorithms supported."

Begin Discussion #3

Raw Public Keys and Pre-Shared Keys

Raw Public Keys and Pre-Shared Keys

What's currently published (CT-12, TS-07) is no longer current.

- An early trusted review identified no need to define new types.
- That existing types (and groupings) could be used:
  - a "raw-public-key" is the same as existing ct:public-key
  - a PSK (pre-shared or pairwise-symmetric) key is the same as ct:symmetric-key

So now it look like this:

```
module: ietf-truststore
  +--rw truststore
     +--rw certificates* [name] {x509-certificates}?
     |  +-- ...
     +--rw host-keys* [name] {ssh-host-keys}?
     |  +-- ...
     +--rw raw-public-keys* [name] {raw-public-keys}?
        +-- ...
```

no PSK because same is necessarily configured in the Keystore

Any thoughts or concerns?
- should these be merged?

```
module: ietf-keystore
  +--rw keystore
     +--rw asymmetric-keys          ← supports raw *private* keys
     |  +-- ...
     +--rw symmetric-keys           ← supports PSKs
        +-- ...
```

RPK and PSK impact on the TLS Model

(RPK/PSK have no impact on the SSH model)

```
grouping tls-client-grouping
 +-- client-identity
 |  +-- (auth-type)
 |     +--:(certificate)
 |     |  +-- certificate {x509-certificate-auth}?
 |     |     +---u ks:local-or-keystore-end-entity-cert-with-key-grouping
 |     +--:(raw-public-key)
 |     |  +-- raw-public-key {raw-public-key-auth}?
 |     |     +---u ks:local-or-keystore-asymmetric-key-grouping
 |     +--:(psk)
 |        +-- psk {psk-auth}?
 |           +---u ks:local-or-keystore-symmetric-key-grouping
 +-- server-authentication
 |  +-- ca-certs! {x509-certificate-auth}?
 |  |  +-- ...
 |  +-- server-certs! {x509-certificate-auth}?
 |  |  +-- ...
 |  +-- raw-public-keys! {raw-public-key-auth}?
 |     +-- ...
 |
```

Only the "client" grouping is shown.
The "server" grouping is almost identical
(just swap "client" and "server" throughout)

RPK and PSK impact on the NC/RC Models?

If clients identify themselves via RPK or PSK...

- how would servers extract a "username"?

*This is a non-issue for clients, as they *know* what server they're connecting to.*

Options:

1. Define the "psk-to-name" and "rpk-to-name" maps now? needed?
2. Leave definition for some future update?

To anyone feeling that we should define now, are you willing to drive the discussion?

Begin Discussion #4

Client Authentication: local-or-external

Client Authentication: local-or-external

In the currently published versions of the SSH and TLS modules:

- there is a "local-or-external" flag to indicate if client-authentication is defined inside or outside of the data model (i.e., where is the list of clients?)

However, recent on-list discussion led to the following:

Removal of the "choice local-or-external" by:

- Removing the 'external' case.
- Flattening the 'local' case.
- Adding a "client-auth-config-supported" feature.

Any objections?

Begin Discussion #5

Client authentication: required-or-optional

Client Authentication: required-or-optional

In the currently published versions of the TLS and HTTP modules:

- there is a "required-or-optional" flag to indicate if client-authentication must succeed at that protocol layer or not.
- this because, e.g., RESTCONF auth is TLS and/or HTTP.

However, recent on-list discussion has led to following:

Removal of the "choice required-or-optional" because:

- Code wasn't using the flag so much as keying-off what credentials had been configured.
- Examples:
  - if a trust anchor has been configured, then TLS-auth MUST succeed
  - if a password has been configured, then HTTP-auth MUST succeed
  - etc.

Any thoughts or concerns?

Begin Discussion #6

Cert-to-name Fingerprints

Cert-to-name Fingerprints

Both the ietf-[net/rest]conf-server modules use the "cert-to-name" grouping to map client-certificates to a NC/RC "username".

This grouping contains a "mandatory true" node call "fingerprint".

However, a very common deployment scenario will have a common strategy (i.e., all certs are signed the same way), in which case there is a common extraction-strategy, and hence no need to specify the fingerprint.

Currently, these modules refine the fingerprint "mandatory false"

- doesn't prevent existing behavior
- should be used with care (as all things with security)

Any thoughts or concerns?

🙏 Thanks for the input! ☺️