Status and Issues for the “Client-Server” Drafts

draft-ietf-netconf-crypto-types-02
draft-ietf-netconf-trust-anchors-03
draft-ietf-netconf-keystore-07
draft-ietf-netconf-ssh-client-server-08
draft-ietf-netconf-tls-client-server-08
draft-ietf-netconf-netconf-client-server-08
draft-ietf-netconf-restconf-client-server-08

NETCONF WG
IETF 103 (Bangkok)

Since IETF 102

All drafts updated and submitted as a set…twice!

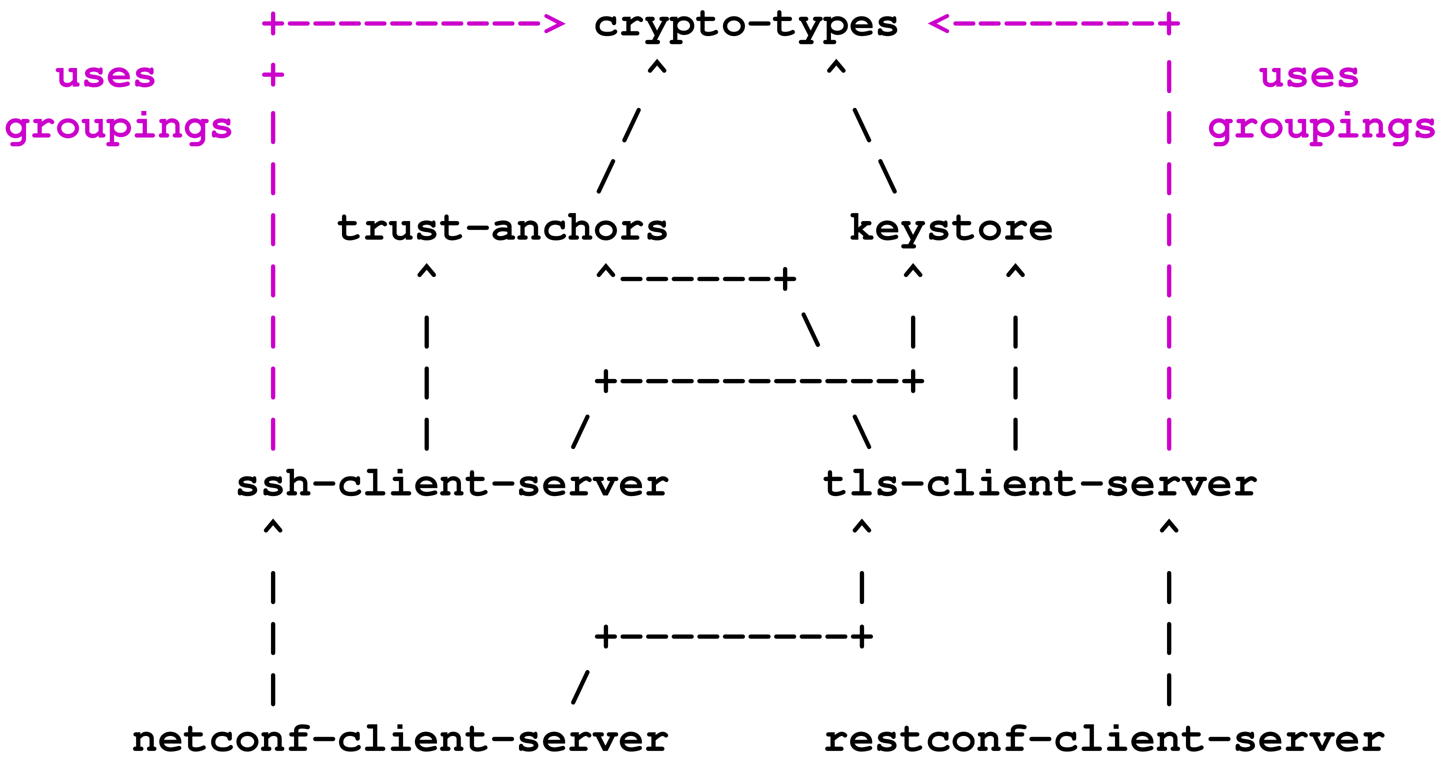
- Most issues discussed in Montreal now resolved.
- A few additional fixes were mode as well

Two issues remain:

1. Should algorithm identities be moved from ietf-[ssh/tls]-common to crypto-types?
2. Add support for TCP Keepalives?

This presentation only focuses on these two issues.

Quick Recap: Relationship between Drafts



```
          +----------> crypto-types <--------+
  uses    +              ^      ^            |   uses
groupings |             /        \           | groupings
          |            /          \          |
          |  trust-anchors        keystore    |
          |     ^      ^------+    ^   ^     |
          |     |              \   |   |     |
          |     |      +-----------+   |     |
          |     |     /          \     |     |
         ssh-client-server      tls-client-server
          ^                      ^           ^
          |                      |           |
          |            +---------+           |
          |           /                      |
     netconf-client-server      restconf-client-server
```

Begin discussion on Issue #1

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

crypto-types updates

- Added many new cryptographic algorithms for completeness.
- New references to:
  - IPSec [RFC8221], IKEv2 [RFC8247], TLS 1.2 [RFC5246], TLS 1.3 [RFC8446], SSH-2 [RFC4253], …
- There are now six categories of crypto algorithms and related identities:
  - **hash-algorithm**: sha1, sha224, …
  - **symmetric-encryption-algorithm**: aes-128-cbc, …, aes-128-ccm, …, aes-128-gcm, …, chacha20-poly1305
  - **mac-algorithm**: hmac-sha1, …, hmac-sha2-512, hamc-sha2-512-256, aes-128-gmac, …, aes-CMAC-96, …, aes-128-ccm, …, aes-128-gcm, …, chacha20-poly1305
  - **asymmetric-encryption-algorithm**: rsa1024, …, rsa15360
  - **signature-algorithm**: rsa-pkcs1-sha1, …, rsa-pss-rsae-sha256, …, rsa-pss-pss-sha256, …, ecdsa_secp256r1_sha256, …, ed25519, ed448, dsa-sha1, x509v3-rsa-pkcs1-sha1, …, x509v3-ecdsa-secp256r1-sha256
  - **key-negotiation-algorithm**: rsa1024, …, rsa15360, psk-only, dhe-ffdhe2048, …, psk-dhe-ffdhe2048, …, (psk-)ecdhe-secp256r1, (psk-) ecdhe-x25519, (psk-) ecdhe-x448, dh-group1-sha1,…, dh-group18-sha512, ecdh-sha2-secp256r1, …

crypto-types issues

1. Need to refine the asymmetric-key-encryption-algorithm definition:
   - current list (rsa1024, …, rsa15360) may be not complete or accurate.
   - More study and discussion is needed here.
2. How to define the key algorithm for public key pair/certificate, based on the six categories of cryptographic algorithms?
   - **Option 1**: A fine-grained way. To be an union of three algorithms (signature, asymmetric encryption and key exchange), with some statements about how these three algorithms can be combined together to represent a valid RSA or ECC plus DH/DHE suite;
   - **Option 2**: A coarse-grained way. Just use a general identifier “key-algorithm” simply
   - **Other options?**

Help is welcomed!!!

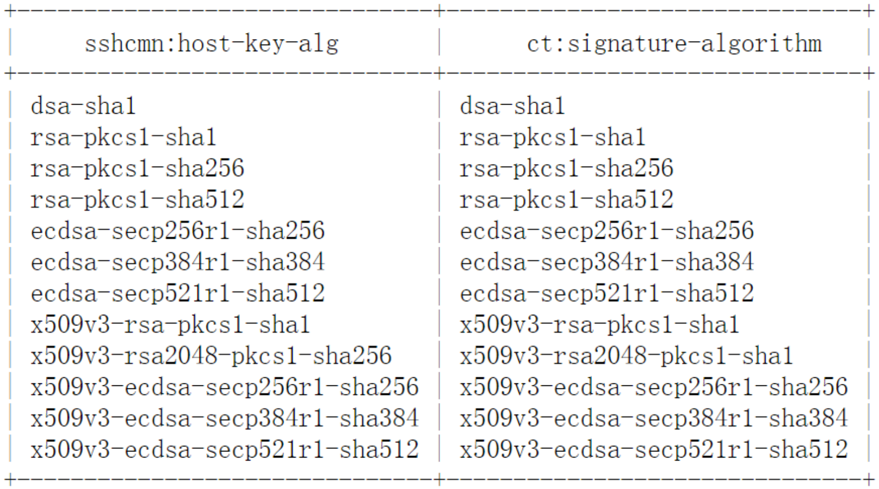
ssh-client-server updates

Section 5 now includes:

- An analysis of cryptographic algorithms of the SSH-2 [RFC4253]. In summary, there are four categories of cryptographic algorithms: host-key-alg, key-exchange-alg, encryption-alg and mac-alg.
- Four compatibility-matrix tables indicate how configured SSH-2 cryptographic algorithm values need to be compatible with the configured private key, having its key algorithm identity defined in crypto-types-02:
  - The SSH-2 Host-key-alg Compatibility Matrix Table
  - The SSH-2 Key-exchange-alg Compatibility Matrix Table
  - The SSH-2 Encryption-alg Compatibility Matrix Table
  - The SSH-2 Mac-alg Compatibility Matrix Table

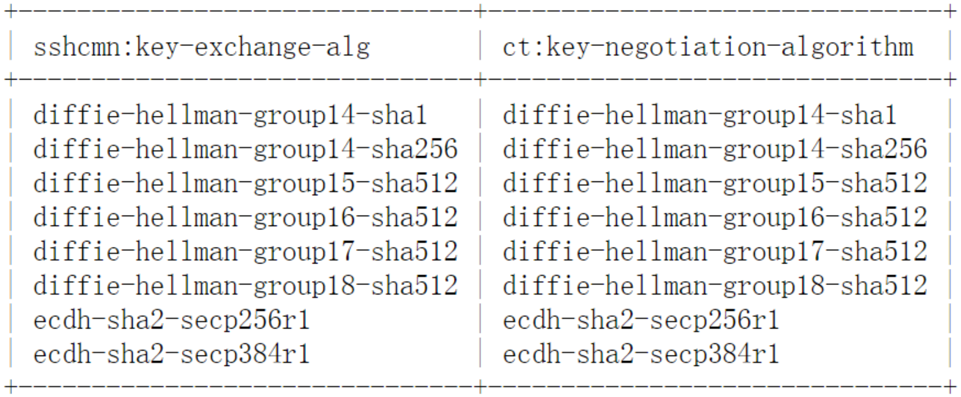
ssh-client-server updates (cont.)

SSH-2 Cryptographic Algorithm Compatibility Matrix Tables



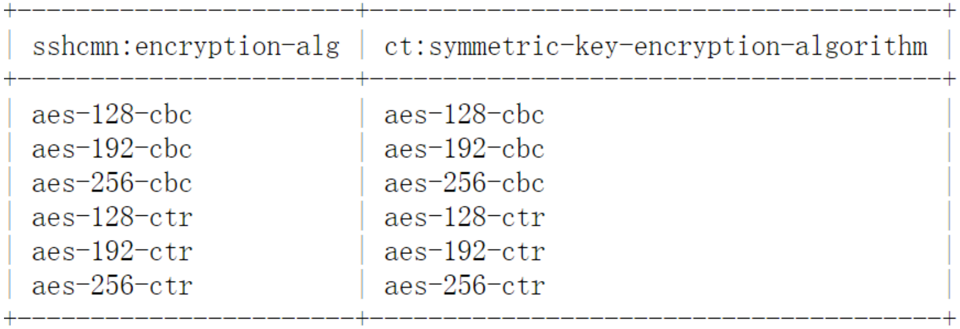
sshcmn:host-key-alg	ct:signature-algorithm
dsa-sha1	dsa-sha1
rsa-pkcs1-sha1	rsa-pkcs1-sha1
rsa-pkcs1-sha256	rsa-pkcs1-sha256
rsa-pkcs1-sha512	rsa-pkcs1-sha512
ecdsa-secp256r1-sha256	ecdsa-secp256r1-sha256
ecdsa-secp384r1-sha384	ecdsa-secp384r1-sha384
ecdsa-secp521r1-sha512	ecdsa-secp521r1-sha512
x509v3-rsa-pkcs1-sha1	x509v3-rsa-pkcs1-sha1
x509v3-rsa2048-pkcs1-sha256	x509v3-rsa2048-pkcs1-sha1
x509v3-ecdsa-secp256r1-sha256	x509v3-ecdsa-secp256r1-sha256
x509v3-ecdsa-secp384r1-sha384	x509v3-ecdsa-secp384r1-sha384
x509v3-ecdsa-secp521r1-sha512	x509v3-ecdsa-secp521r1-sha512

Table 1 The SSH Host-key-alg Compatibility Matrix



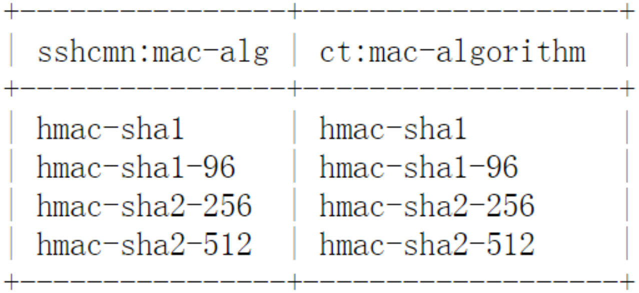
sshcmn:key-exchange-alg	ct:key-negotiation-algorithm
diffie-hellman-group14-sha1	diffie-hellman-group14-sha1
diffie-hellman-group14-sha256	diffie-hellman-group14-sha256
diffie-hellman-group15-sha512	diffie-hellman-group15-sha512
diffie-hellman-group16-sha512	diffie-hellman-group16-sha512
diffie-hellman-group17-sha512	diffie-hellman-group17-sha512
diffie-hellman-group18-sha512	diffie-hellman-group18-sha512
ecdh-sha2-secp256r1	ecdh-sha2-secp256r1
ecdh-sha2-secp384r1	ecdh-sha2-secp384r1

Table 2 The SSH Key-exchange-alg Compatibility Matrix



sshcmn:encryption-alg	ct:symmetric-key-encryption-algorithm
aes-128-cbc	aes-128-cbc
aes-192-cbc	aes-192-cbc
aes-256-cbc	aes-256-cbc
aes-128-ctr	aes-128-ctr
aes-192-ctr	aes-192-ctr
aes-256-ctr	aes-256-ctr

Table 3 The SSH Encryption-alg Compatibility Matrix



sshcmn:mac-alg	ct:mac-algorithm
hmac-sha1	hmac-sha1
hmac-sha1-96	hmac-sha1-96
hmac-sha2-256	hmac-sha2-256
hmac-sha2-512	hmac-sha2-512

Table 4 The SSH Mac-alg Compatibility Matrix

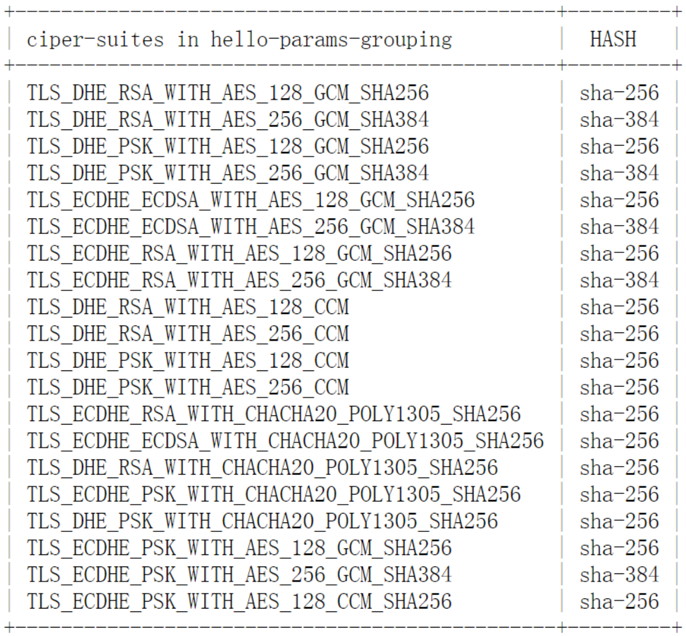
tls-client-server updates

Section 5 now includes

- An analysis of cryptographic algorithms of the TLS 1.2 and TLS 1.3
    - For TLS1.2, there are 4 categories of cryptographic algorithms: TLS Cipher Suites, TLS SignatureAlgorithm, TLS HashAlgorithm, TLS Supported Groups
    - For TLS 1.3, there are 3 categories of cryptographic algorithms: TLS Cipher Suites, TLS SignatureScheme, TLS Supported Groups
- Compatibility-matrix tables indicate how configured "host-key-alg" values of TLS need to be compatible with the configured private key, having its key algorithm identity defined in crypto-types-02
    - For TLS 1.2, add 5 tables: TLS ciper suites mapping to hash-algorithm, symmetric-key-encryption-algorithm, mac-algorithm, signature-algorithm, key-negotiation-algorithm
    - For TLS 1.3, add 5 tables: TLS ciper suites mapping to hash-algorithm, symmetric-key-encryption-algorithm, mac-algorithm; SignatureScheme mapping to signature-algorithm; Supported Groups mapping to key-negotiation-algorithm

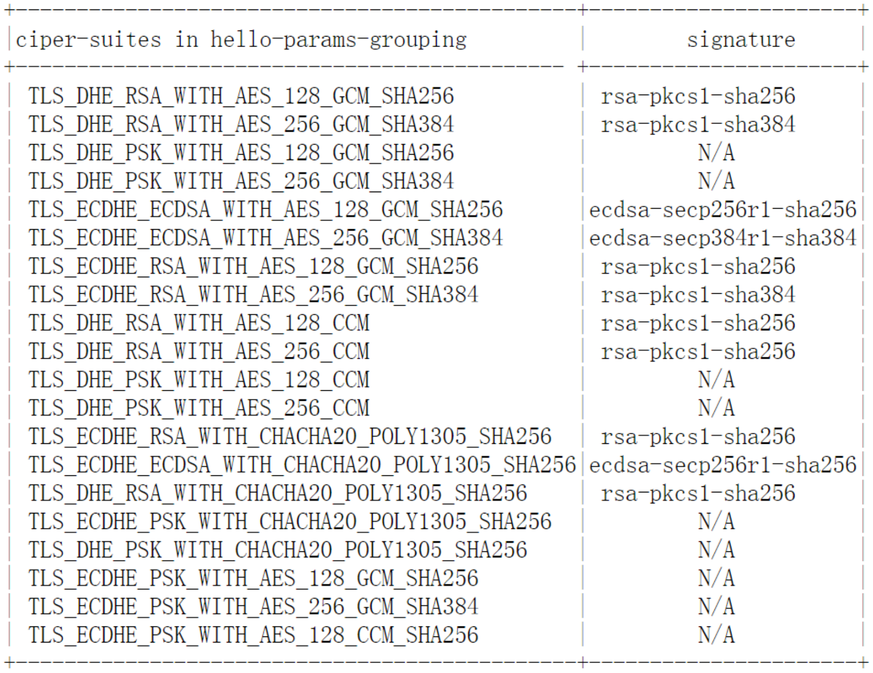
tls-client-server updates (cont.)

TLS 1.2 Cryptographic Algorithm Compatibility Matrix Tables



ciper-suites in hello-params-grouping	HASH
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	sha-256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	sha-384
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	sha-256
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	sha-384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	sha-256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	sha-384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	sha-256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	sha-384
TLS_DHE_RSA_WITH_AES_128_CCM	sha-256
TLS_DHE_RSA_WITH_AES_256_CCM	sha-256
TLS_DHE_PSK_WITH_AES_128_CCM	sha-256
TLS_DHE_PSK_WITH_AES_256_CCM	sha-256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	sha-256
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	sha-256
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	sha-384
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	sha-256

Table 1-1 TLS 1.2 Compatibility Matrix Part 1: ciper-suites mapping to hash-algorithm

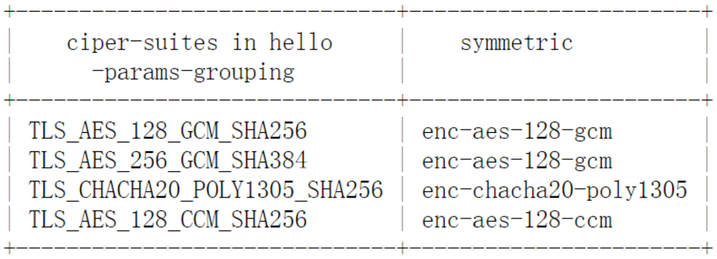


ciper-suites in hello-params-grouping	signature
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	rsa-pkcs1-sha256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	rsa-pkcs1-sha384
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	N/A
TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	N/A
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ecdsa-secp256r1-sha256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ecdsa-secp384r1-sha384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	rsa-pkcs1-sha256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	rsa-pkcs1-sha384
TLS_DHE_RSA_WITH_AES_128_CCM	rsa-pkcs1-sha256
TLS_DHE_RSA_WITH_AES_256_CCM	rsa-pkcs1-sha256
TLS_DHE_PSK_WITH_AES_128_CCM	N/A
TLS_DHE_PSK_WITH_AES_256_CCM	N/A
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	rsa-pkcs1-sha256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	ecdsa-secp256r1-sha256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	rsa-pkcs1-sha256
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	N/A
TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	N/A
TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	N/A
TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	N/A
TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	N/A

Table 1-4 TLS 1.2 Compatibility Matrix Part 4: ciper-suites mapping to signature-algorithm

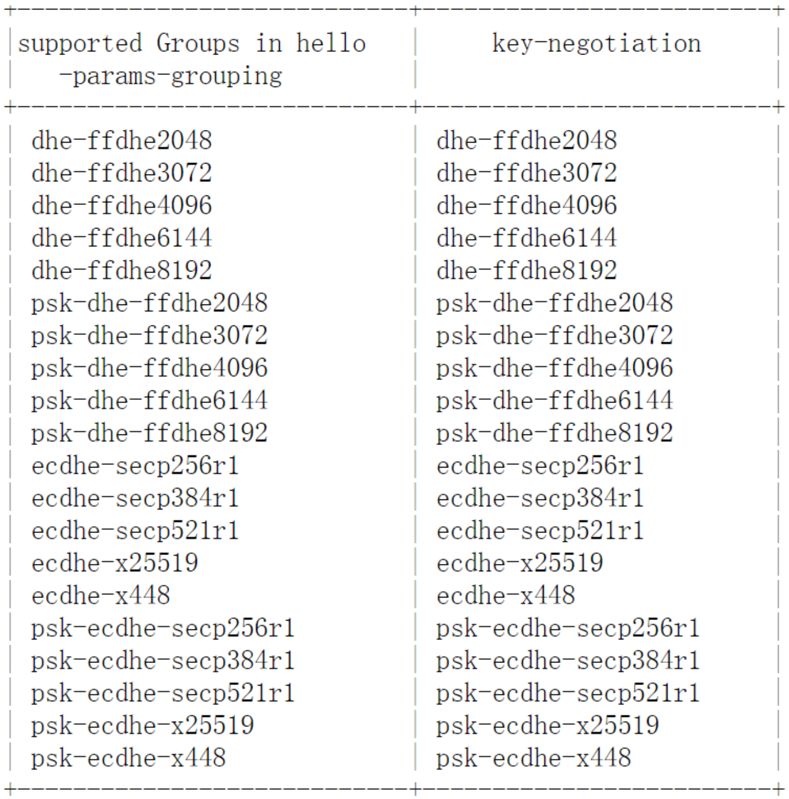
tls-client-server updates (cont.)

TLS 1.3 Cryptographic Algorithm Compatibility Matrix Tables



ciper-suites in hello -params-grouping	symmetric
TLS_AES_128_GCM_SHA256	enc-aes-128-gcm
TLS_AES_256_GCM_SHA384	enc-aes-128-gcm
TLS_CHACHA20_POLY1305_SHA256	enc-chacha20-poly1305
TLS_AES_128_CCM_SHA256	enc-aes-128-ccm

Table 2-2 TLS 1.3 Compatibility Matrix Part 2: ciper-suites mapping to symmetric-key--encryption-algorithm



supported Groups in hello -params-grouping	key-negotiation
dhe-ffdhe2048	dhe-ffdhe2048
dhe-ffdhe3072	dhe-ffdhe3072
dhe-ffdhe4096	dhe-ffdhe4096
dhe-ffdhe6144	dhe-ffdhe6144
dhe-ffdhe8192	dhe-ffdhe8192
psk-dhe-ffdhe2048	psk-dhe-ffdhe2048
psk-dhe-ffdhe3072	psk-dhe-ffdhe3072
psk-dhe-ffdhe4096	psk-dhe-ffdhe4096
psk-dhe-ffdhe6144	psk-dhe-ffdhe6144
psk-dhe-ffdhe8192	psk-dhe-ffdhe8192
ecdhe-secp256r1	ecdhe-secp256r1
ecdhe-secp384r1	ecdhe-secp384r1
ecdhe-secp521r1	ecdhe-secp521r1
ecdhe-x25519	ecdhe-x25519
ecdhe-x448	ecdhe-x448
psk-ecdhe-secp256r1	psk-ecdhe-secp256r1
psk-ecdhe-secp384r1	psk-ecdhe-secp384r1
psk-ecdhe-secp521r1	psk-ecdhe-secp521r1
psk-ecdhe-x25519	psk-ecdhe-x25519
psk-ecdhe-x448	psk-ecdhe-x448

Table 2-5 TLS 1.3 Compatibility Matrix Part 5: Supported Groups mapping to key-negotiation-algorithm

Begin discussion on Issue #2

Add support for TCP Keepalives?

Last time we discussed how discussions with the Transport Area folks concluded that there is a need for keepalives at every protocol layer (TCP, SSH, TLS, NETCONF, RESTCONF, etc.)

- Aliveness of a lower layer says nothing about the aliveness of an upper layer
- Aliveness checks at an upper layer SHOULD NOT not preclude aliveness checks at a lower layer.

The question we're stuck on is *how* to configured keepalives at the various layers…

An idea, but you may not like it…

Independent of this discussion, I’ve been aware of gaps in our current solution. Specifically that we’re missing the dependent protocol layers: TCP, HTTP, and HTTPS.

FWIW: I withheld raising this to the WG because I envisioned much eye-rolling and general exasperation, but now it seems that the time has come…

Draft Restructuring Idea

Adding in the missing tcp/http/https-client-server Layers

```
     +----------> crypto-types <-----+
     |              ^      ^         |
     |             /        \        |
     |            /          \       |
     |  trust-anchors      keystore  |   tcp-client-server
     |     ^      ^         ^   ^    |     ^   ^     ^
     |     |      +----+    |   |    +    /   /      |
     |     |            \   |   |   /    /   /       |
     |     |      +---------+   |  +    /   /        |
     |     |     /        \     |  |   /   /         |
     |     |    /   +-----------------+   /          |
     |     |   /   /        \   |  |     /           |
    ssh-client-server     tls-client-server    http-client-server
     ^                      ^          ^             ^
     |                      |          |             |
     |            +---------+        https-client-server
     |           /                            ^
netconf-client-server                         |
                                   restconf-client-server
```

Benefits of Restructuring

Factoring out these dependent layers will provide a basis for future protocols models.

- Surely there will be more TCP-based models.
- Surely there will be more HTTP-based models.
- Surely there will be more HTTPS-based models.

And, back to the Keepalive issue…

Distinct layers enable keepalives to be configured at each layer.

The configuration of:

- TCP-keepalives can be defined in the tcp-client-server models
- SSH-keepalives can be defined in the ssh-client-server models
- TLS-keepalives can be defined in the tls-client-server models
- HTTP-keepalives can be defined in the http-client-server models
- NETCONF-keepalives (TBD) can be defined in the netconf-client-server models
- RESTCONF-keepalives (TBD) can be defined in the restconf-client-server models

Should we do it?

� Thanks for the input! ㅆ