STORAGE DEVELOPER CONFERENCE
SDC 22
Fremont, CA
September 12-15, 2022
BY Developers FOR Developers

A SNIA Event

Scaling NVMe over IP Fabric Security

Claudio DeSanti
Distinguished Engineer
Dell Technologies CTIO Group

Agenda

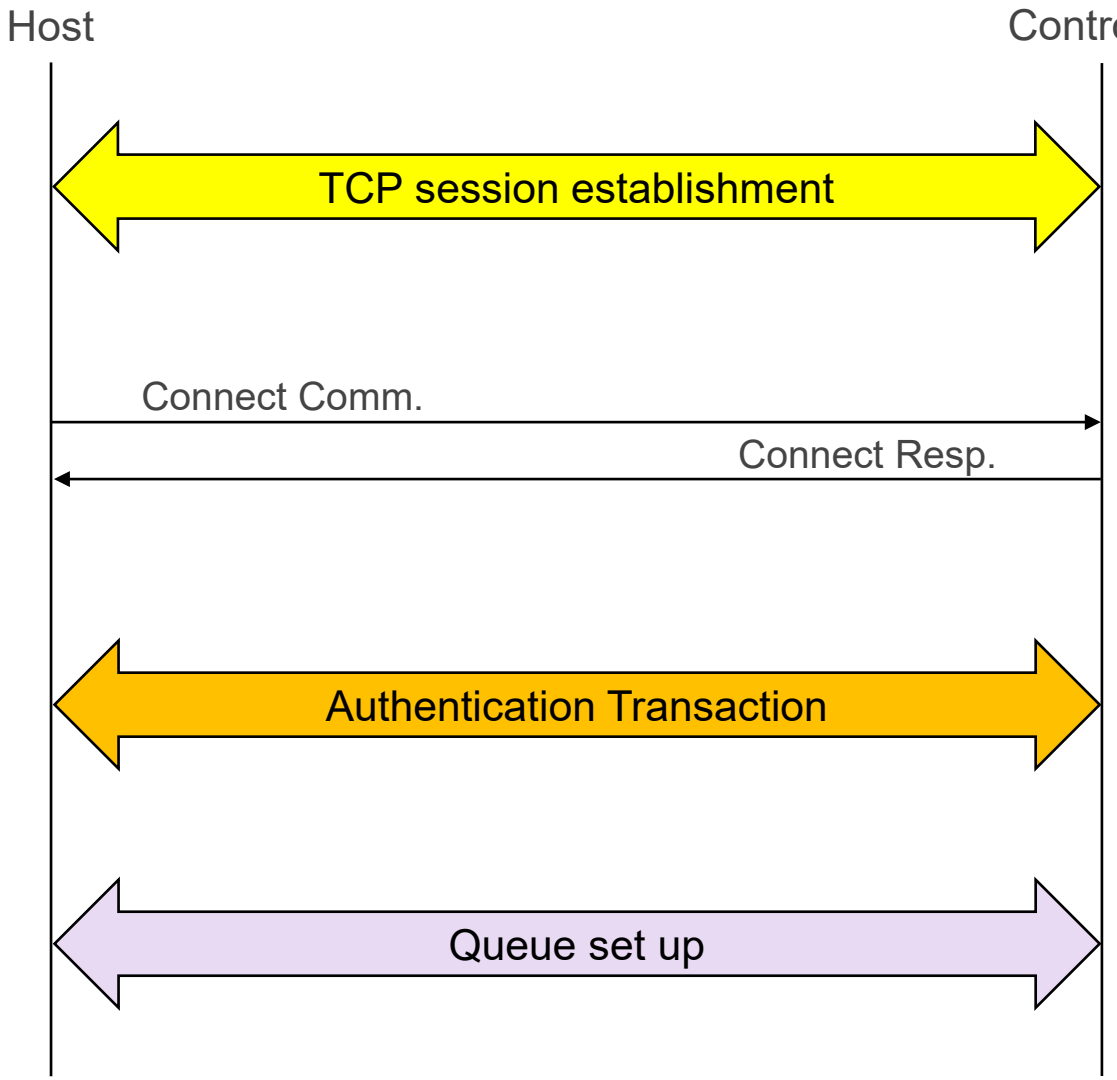
- NVMe Security Functions
- DH-HMAC-CHAP Authentication
- Authentication Verification Entity (AVE) for DH-HMAC-CHAP
- AVE Access and DH-HMAC-CHAP Provisioning

NVMe Security Functions

NVMe Security Functions

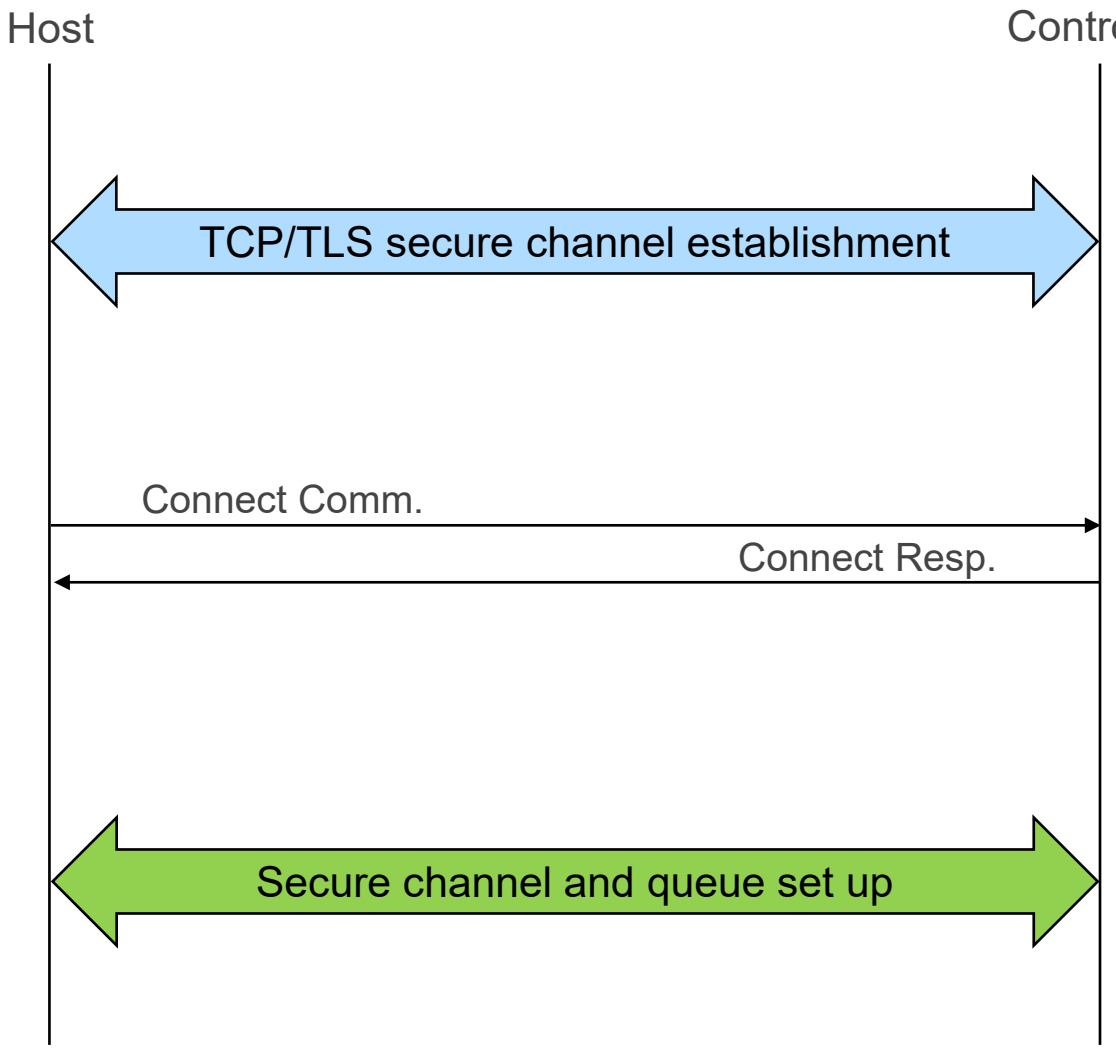
- Storage security requires two fundamental functions:
  - Authentication (i.e., proof of identity of an NVMe entity)
  - Secure channel (i.e., integrity and confidentiality of data in flight)
- NVMe defined the protocols for these functions
  - Authentication: DH-HMAC-CHAP
    - In TP 8006, now in NVMe base spec v2.0
  - Secure channel: TLS 1.3
    - In TP 8011, now in NVMe TCP Transport spec v1.0
- How to deploy at scale these protocols?

NVMe-oF Authentication Example



1. A TCP session is established
2. The Connect exchange is performed to set up NVMe queue and associate host to controller
3. The host performs an authentication transaction with the controller to authenticate the end-points
4. Queue is ready for subsequent operations

TLS 1.3



1. A TCP/TLS session negotiation is performed and a secure channel is established

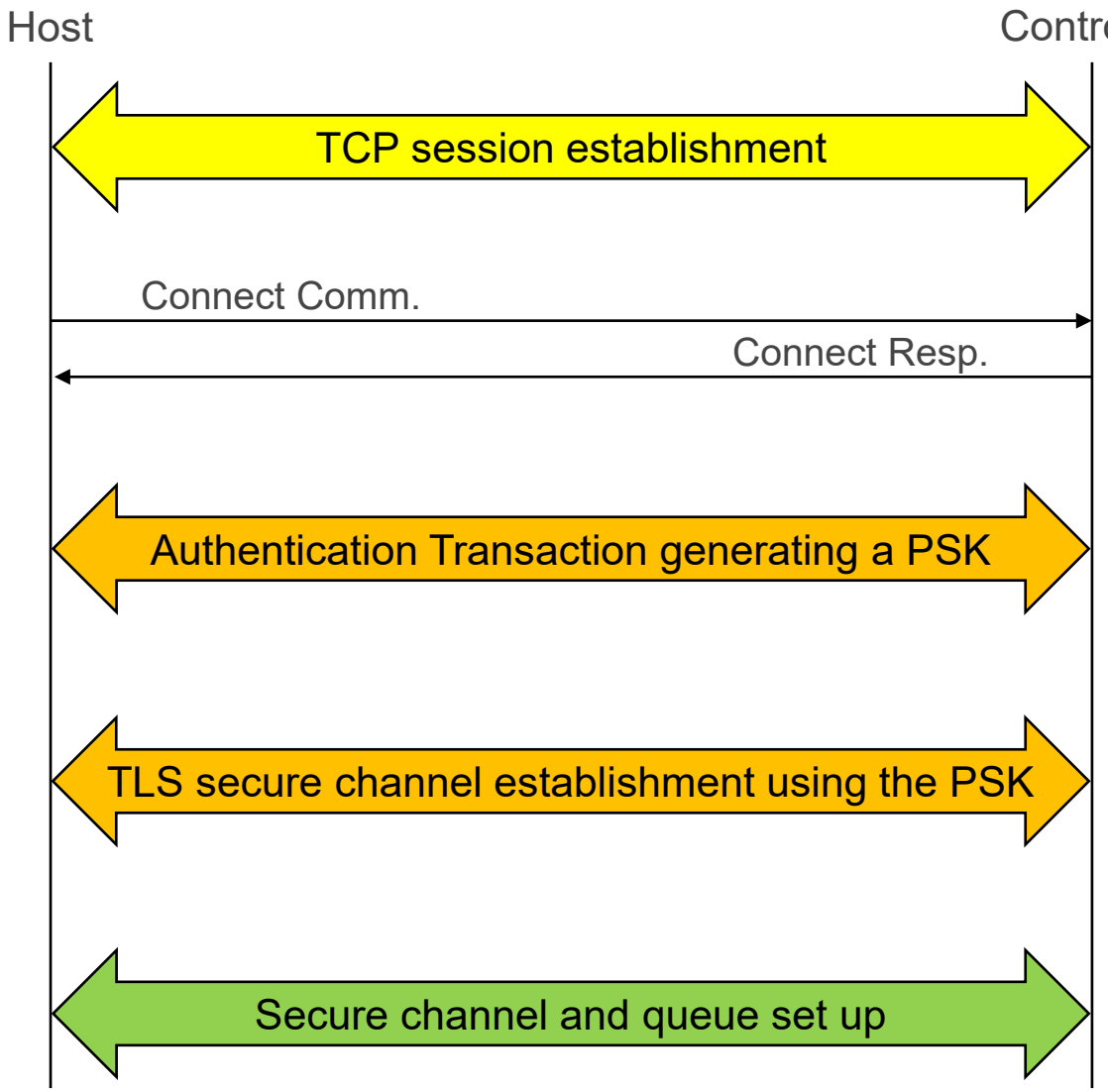
2. The Connect exchange is performed to set up NVMe queue and associate host to controller

3. Secure channel and queue are set up, ready for subsequent operations

TLS Credentials

- TLS secure channel for NVMe/TCP is based on pre-shared keys (PSKs)
  - In order to authenticate and establish a secure channel between themselves, two NVMe entities need to know the same PSK
  - Each pair of entities require its own PSK
    - n^2 problem: a fabric with N hosts and M subsystems require N x M PSKs
- Authentication protocols to the rescue
  - Upon successful completion of an authentication exchange, the two involved NVMe entities generate an ephemeral shared session key (i.e., a PSK computed on the fly)
  - The TLS negotiation can then be performed using a PSK derived from that shared key
  - Implementation result: the TCP connection begins unsecured and then transitions to secured
    - Opportunistic TLS
  - Reduces the TLS provisioning problem to the DH-HMAC-CHAP provisioning problem

TLS Concatenation



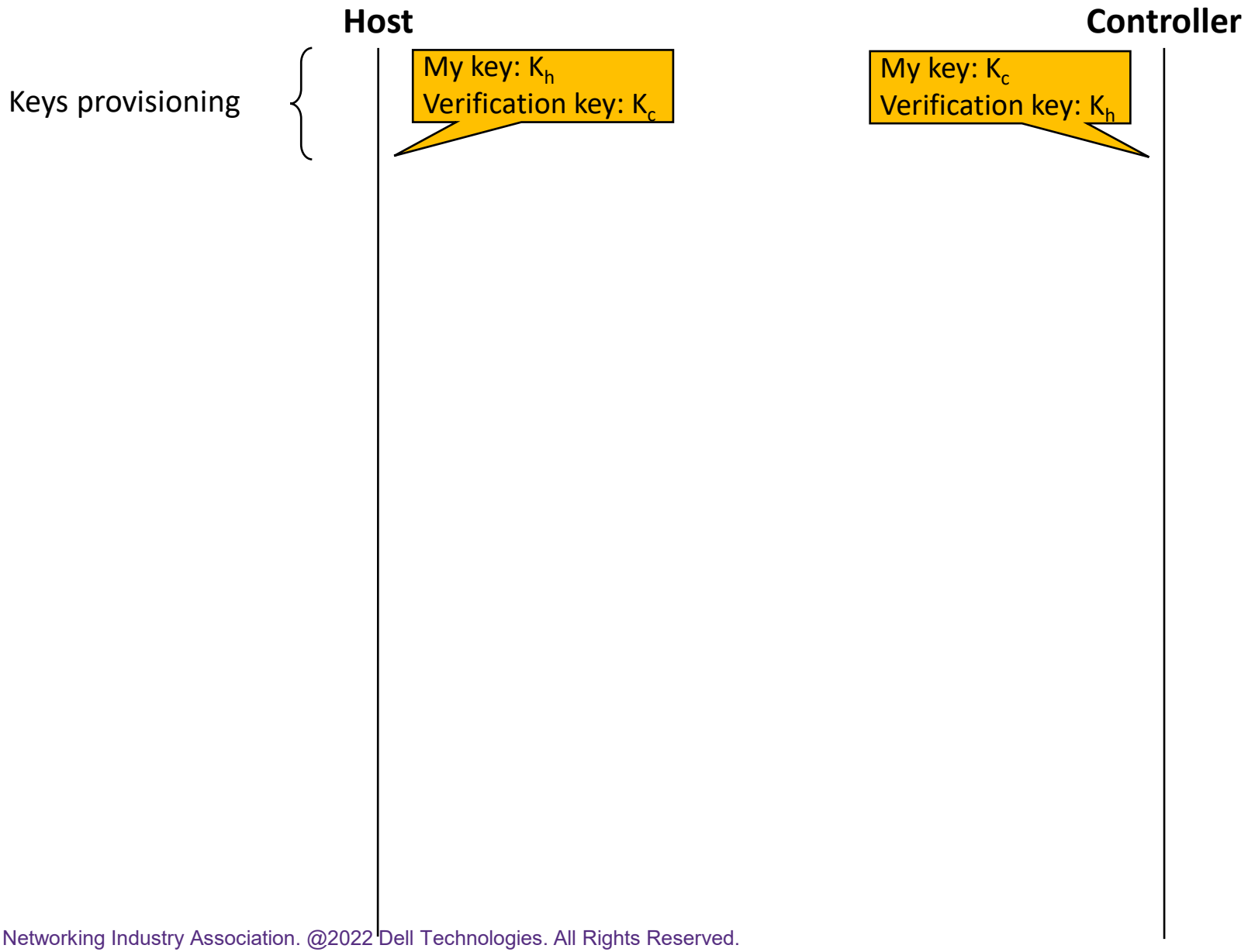
1. A TCP session is established
2. The Connect exchange is performed to set up NVMe queue and associate host to controller
3. The host performs an authentication transaction with the controller, transaction that generates a pre-shared key PSK between host and controller
4. The pre-shared key PSK is used to perform a TLS negotiation and to establish a secure channel
5. Secure channel and queue are set up, ready for subsequent operations

DH-HMAC-CHAP Authentication

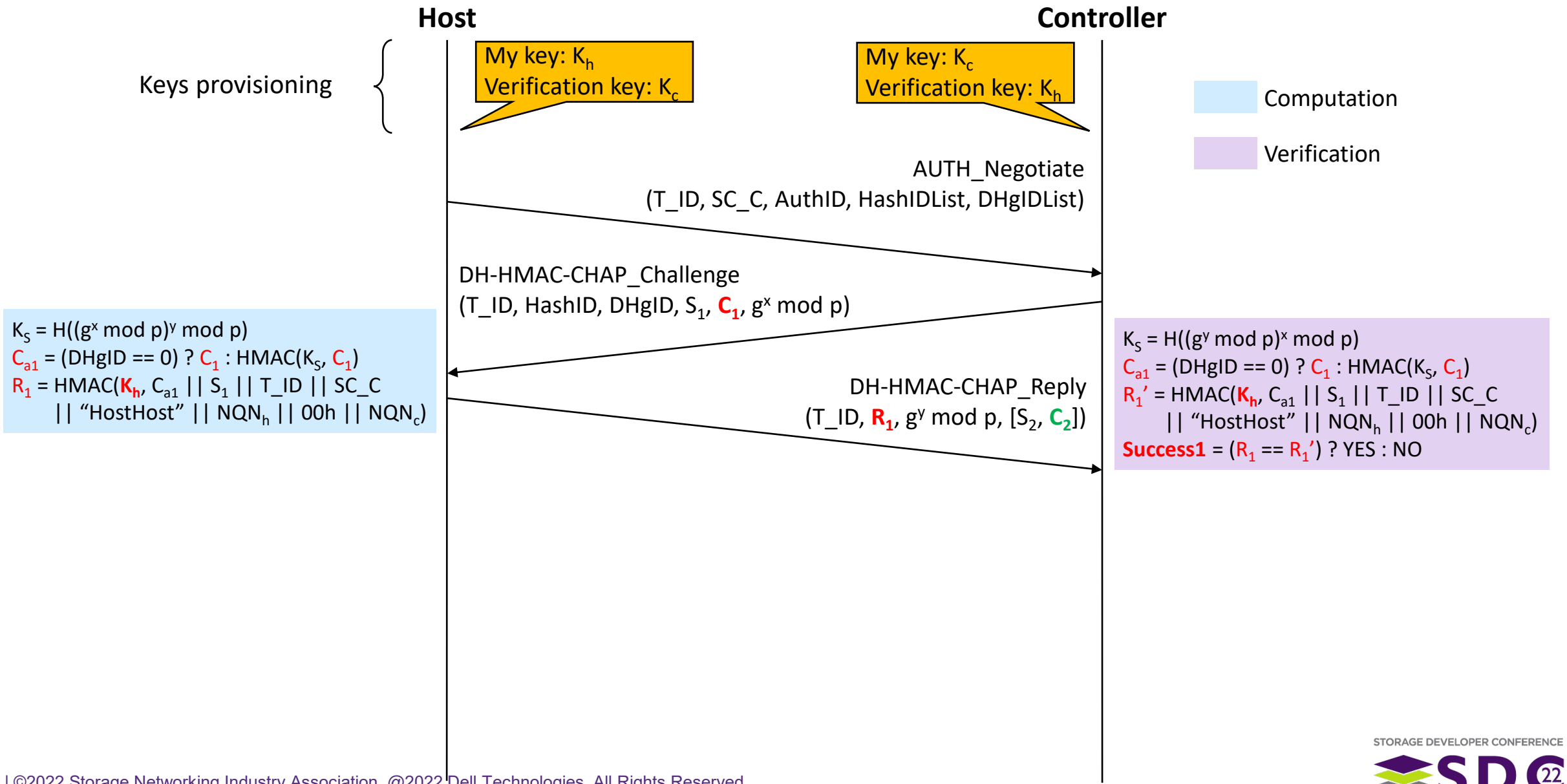
DH-HMAC-CHAP Authentication

- An NVMe entity (Host or Subsystem) is provisioned with a key
- DH-HMAC-CHAP authentication: demonstrate that you know your key
  - A controller authenticates a host if the host demonstrates it knows its own key (K_h)
  - A host authenticates a controller if the controller demonstrates it knows its own key (K_c)
- DH-HMAC-CHAP demonstration of key knowledge: challenge/response protocol
  - The key is not passed in the protocol messages
  - The authenticator sends a challenge C
  - The responder computes a response using its own key
    - i.e., $R = HMAC(key_{responder}, C \,||\, other\ things)$
  - The authenticator verifies the response
- DH-HMAC-CHAP response verification: requires the key of the responder
  - The authenticator computes the expected response R'
  - And verifies if the received R is equal to the computed R'

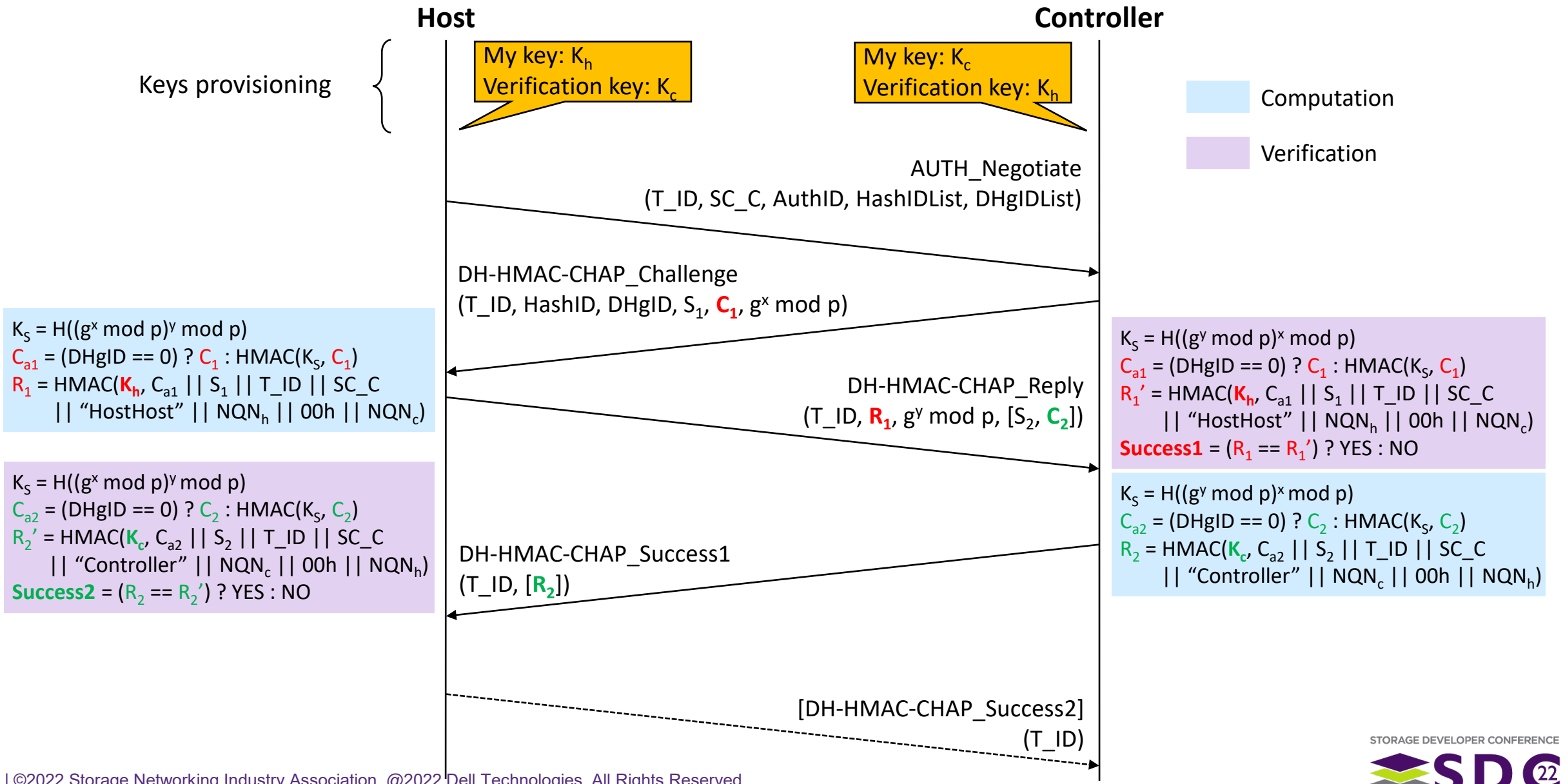
DH-HMAC-CHAP (1)



DH-HMAC-CHAP (2)

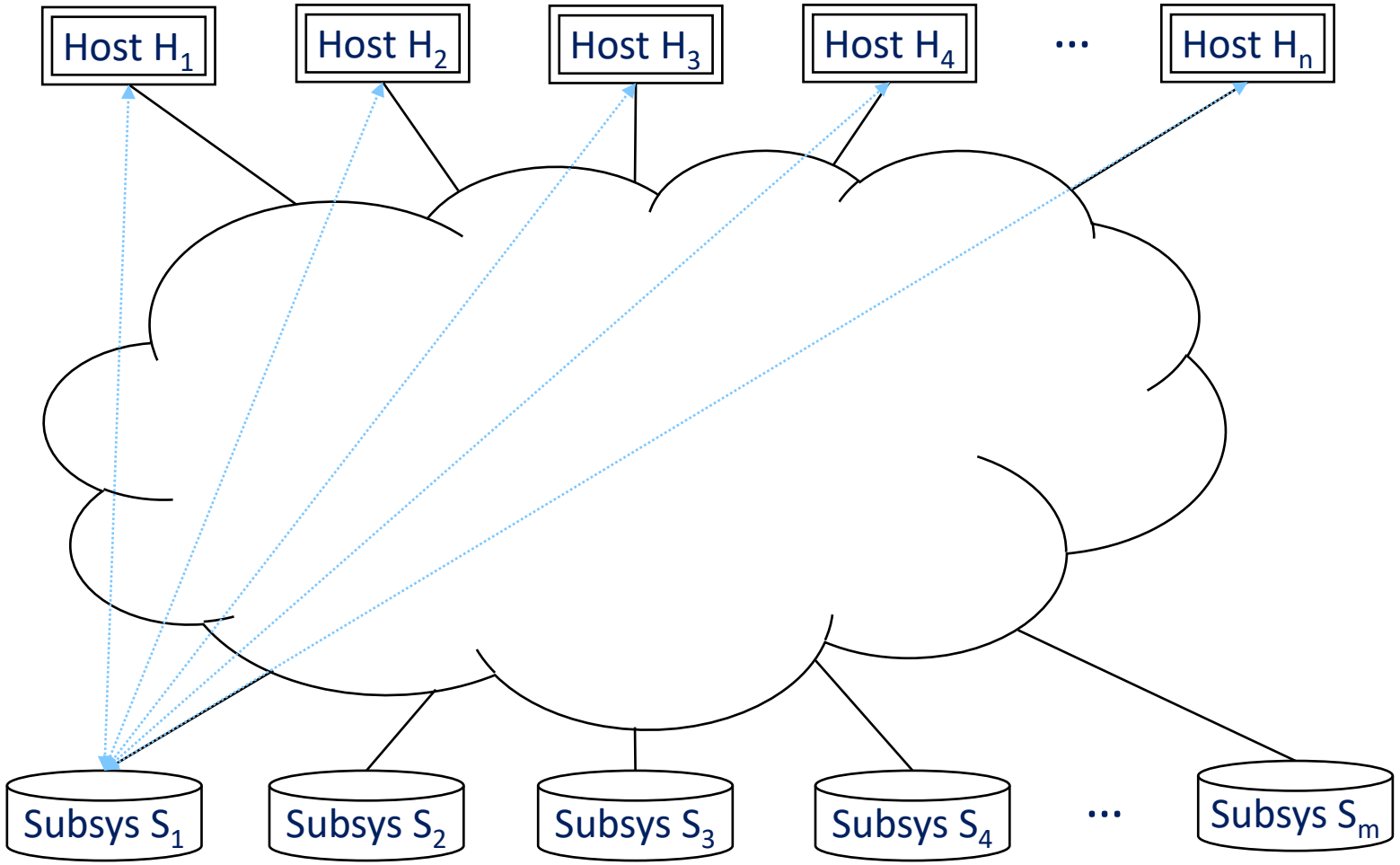


DH-HMAC-CHAP (3)



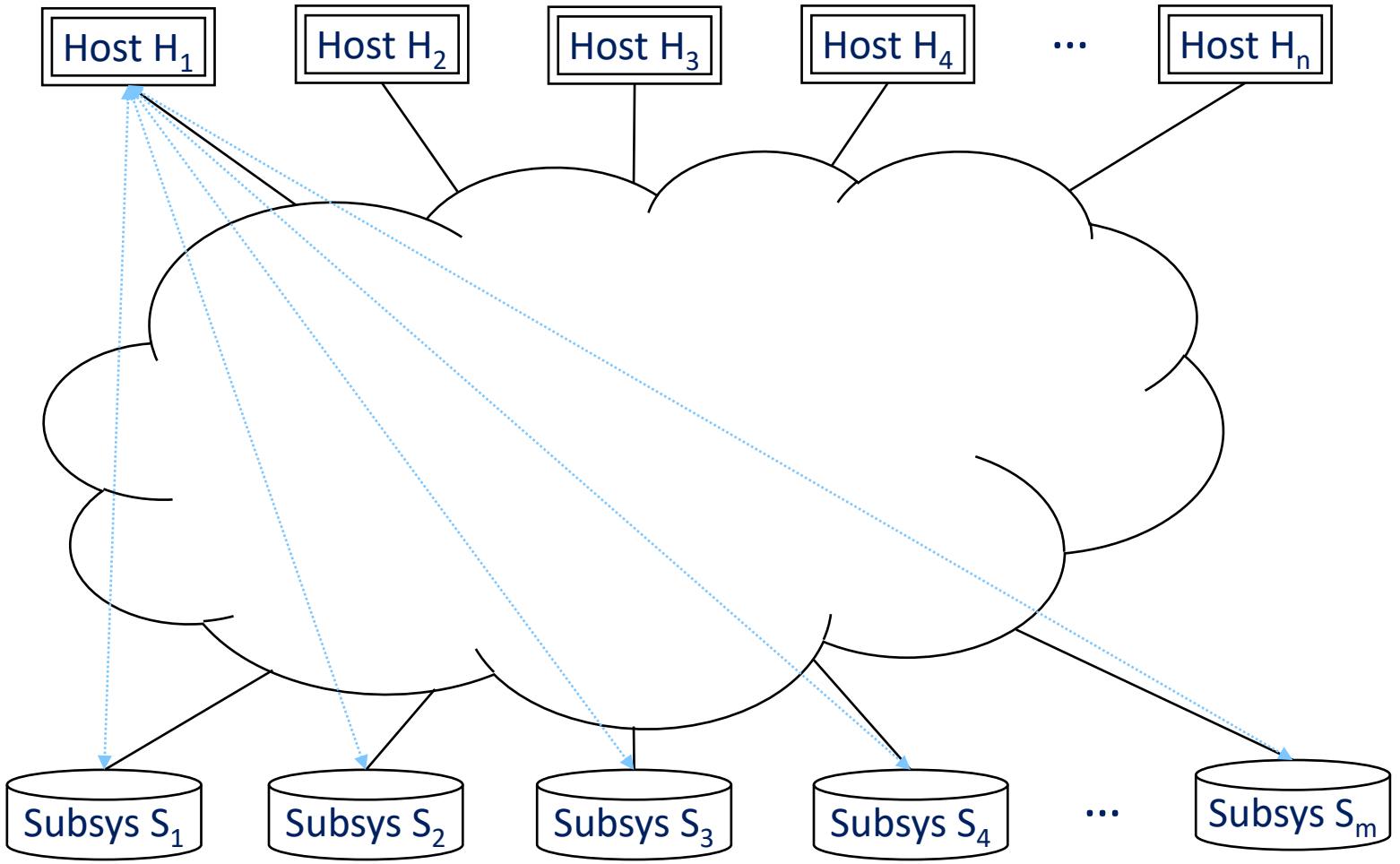
Subsystem Provisioning

- For subsystem S_1:
  - Computation key: K_{S1}
  - Host verification keys:
    - $\{NQN_{H1}, K_{H1}\}$
    - $\{NQN_{H2}, K_{H2}\}$
    - $\{NQN_{H3}, K_{H3}\}$
    - $\{NQN_{H4}, K_{H4}\}$
    - …
    - $\{NQN_{Hn}, K_{Hn}\}$
- Similar configuration for other subsystems



Host Provisioning

- For host H_1:
  - Computation key: K_{H1}
  - Subsystem verification keys:
    - $\{NQN_{S1}, K_{S1}\}$
    - $\{NQN_{S2}, K_{S2}\}$
    - $\{NQN_{S3}, K_{S3}\}$
    - $\{NQN_{S4}, K_{S4}\}$
    - …
    - $\{NQN_{Sm}, K_{Sm}\}$
- Similar configuration for other hosts



DH-HMAC-CHAP Provisioning Scaling Example

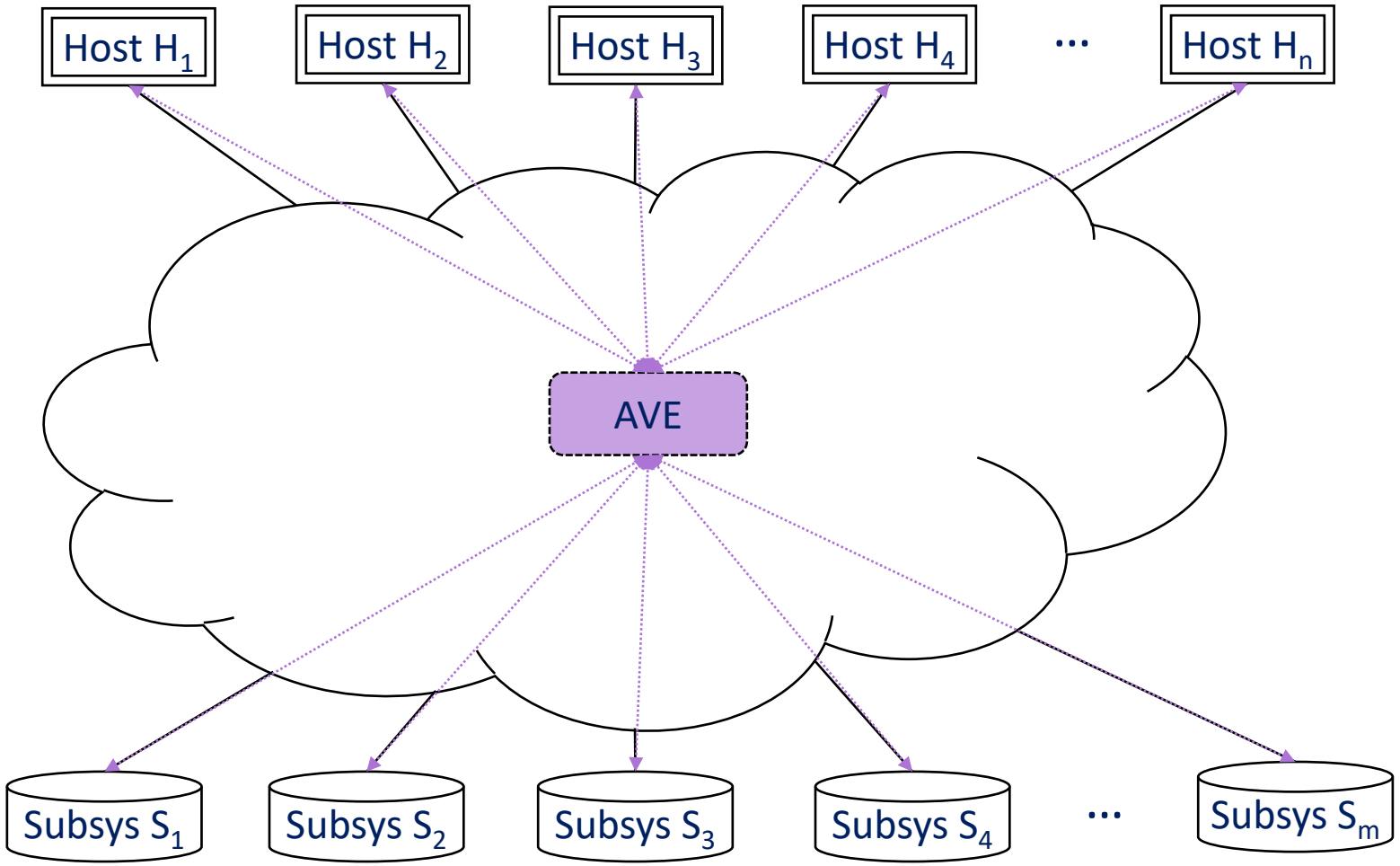
- Let's consider a fabric with 100 hosts and 25 subsystems
  - Suppose all hosts want to authenticate all subsystems
- Initial host provisioning:
  - Each host needs to be provisioned with its own key and the 25 subsystem verification keys
- Subsystem addition to the fabric:
  - All 100 hosts needs to be re-provisioned to add the additional subsystem verification key
- Initial subsystem provisioning:
  - Each subsystem needs to be provisioned with its own key and the 100 host verification keys
- Host addition to the fabric:
  - All 25 subsystems needs to be re-provisioned to add the additional host verification key
- In other words: plain DH-HMAC-CHAP authentication is difficult to deploy

Authentication Verification Entity (AVE) for DH-HMAC-CHAP

Authentication Verification Entity (AVE)

- An offload of the verification processing through a complete database of {NQN, Key} records
  - $\{NQN_{H1}, K_{H1}\}$
  - $\{NQN_{H2}, K_{H2}\}$
  - $\{NQN_{H3}, K_{H3}\}$
  - $\{NQN_{H4}, K_{H4}\}$
  - …
  - $\{NQN_{Hn}, K_{Hn}\}$

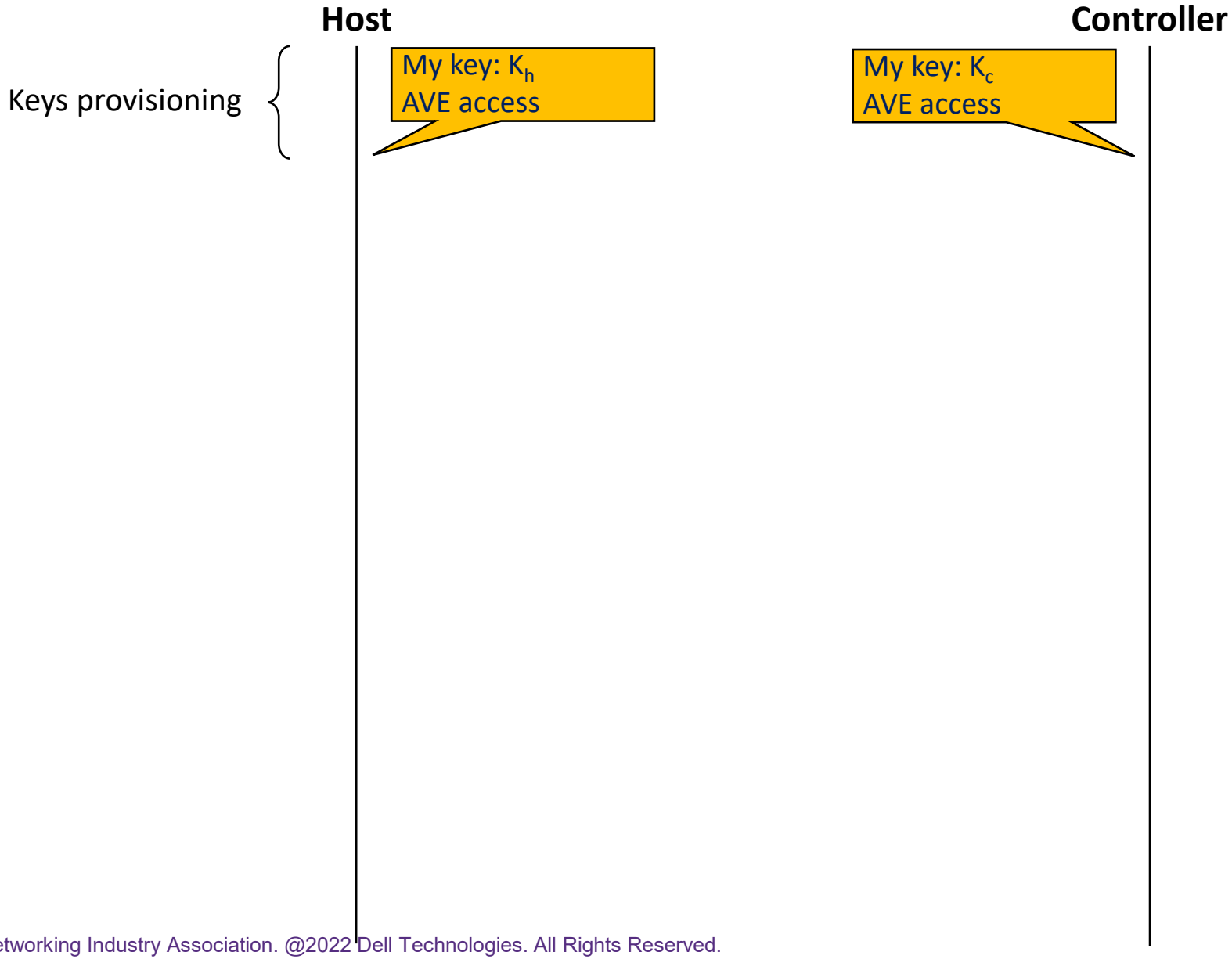
  - $\{NQN_{S1}, K_{S1}\}$
  - $\{NQN_{S2}, K_{S2}\}$
  - $\{NQN_{S3}, K_{S3}\}$
  - $\{NQN_{S4}, K_{S4}\}$
  - …
  - $\{NQN_{Sm}, K_{Sm}\}$



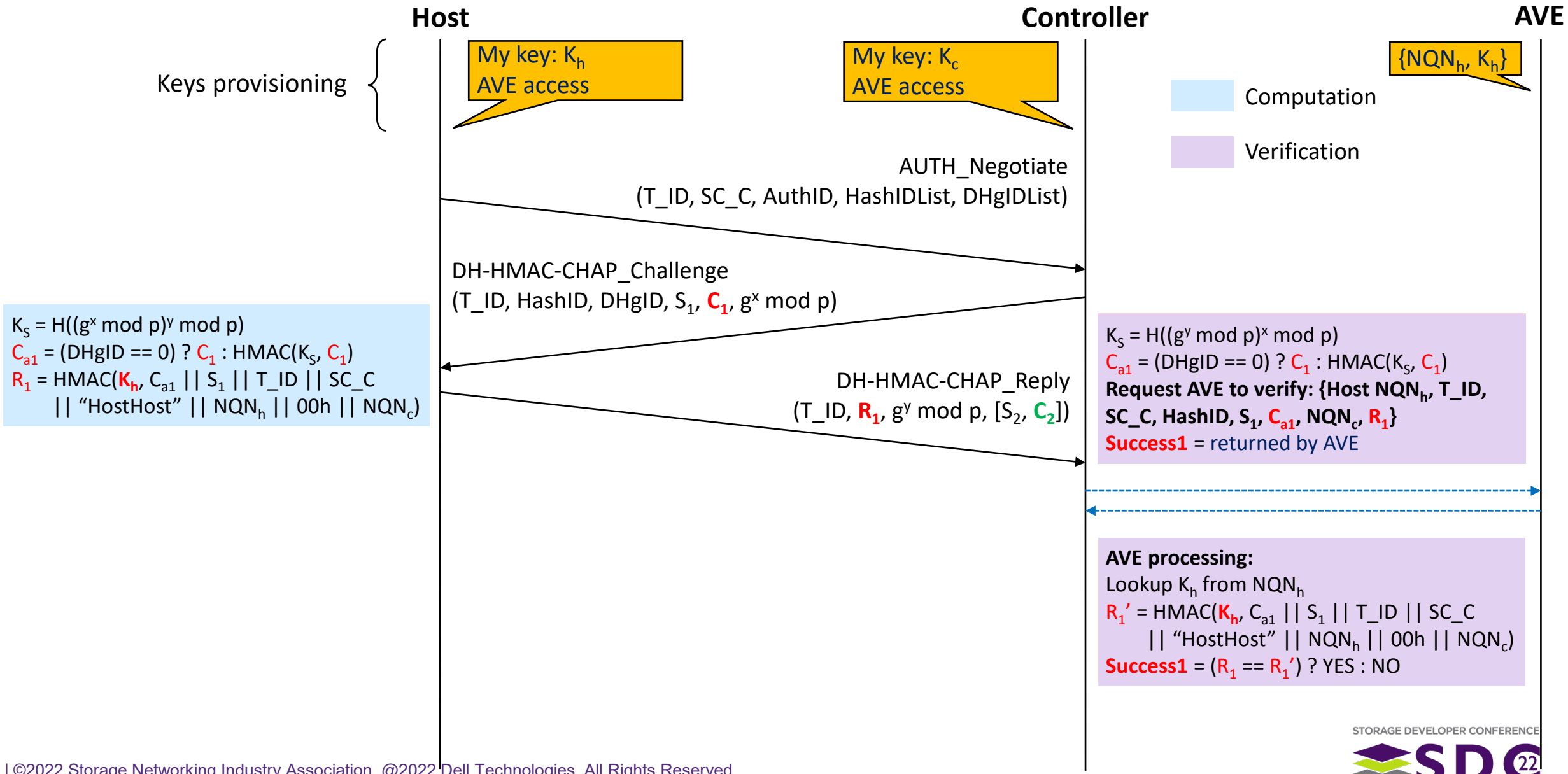
DH-HMAC-CHAP Authentication with AVE

- An NVMe entity (Host or Subsystem) and the AVE are provisioned with a key
- DH-HMAC-CHAP authentication: demonstrate that you know your key
  - A controller authenticates a host if the host demonstrates it knows its own key (K_h)
  - A host authenticates a controller if the controller demonstrates it knows its own key (K_c)
- DH-HMAC-CHAP demonstration of key knowledge: challenge/response protocol
  - The key is not passed in the protocol messages
  - The authenticator sends a challenge C
  - The responder computes a response using its own key
    - i.e., R = HMAC($key_{responder}$, C || other things)
  - The authenticator verifies the response
- DH-HMAC-CHAP response verification: delegated to the AVE
  - Consistent with the RADIUS model for iSCSI CHAP authentication
  - Requires secure access to the AVE
    - E.g., TLS with PSK

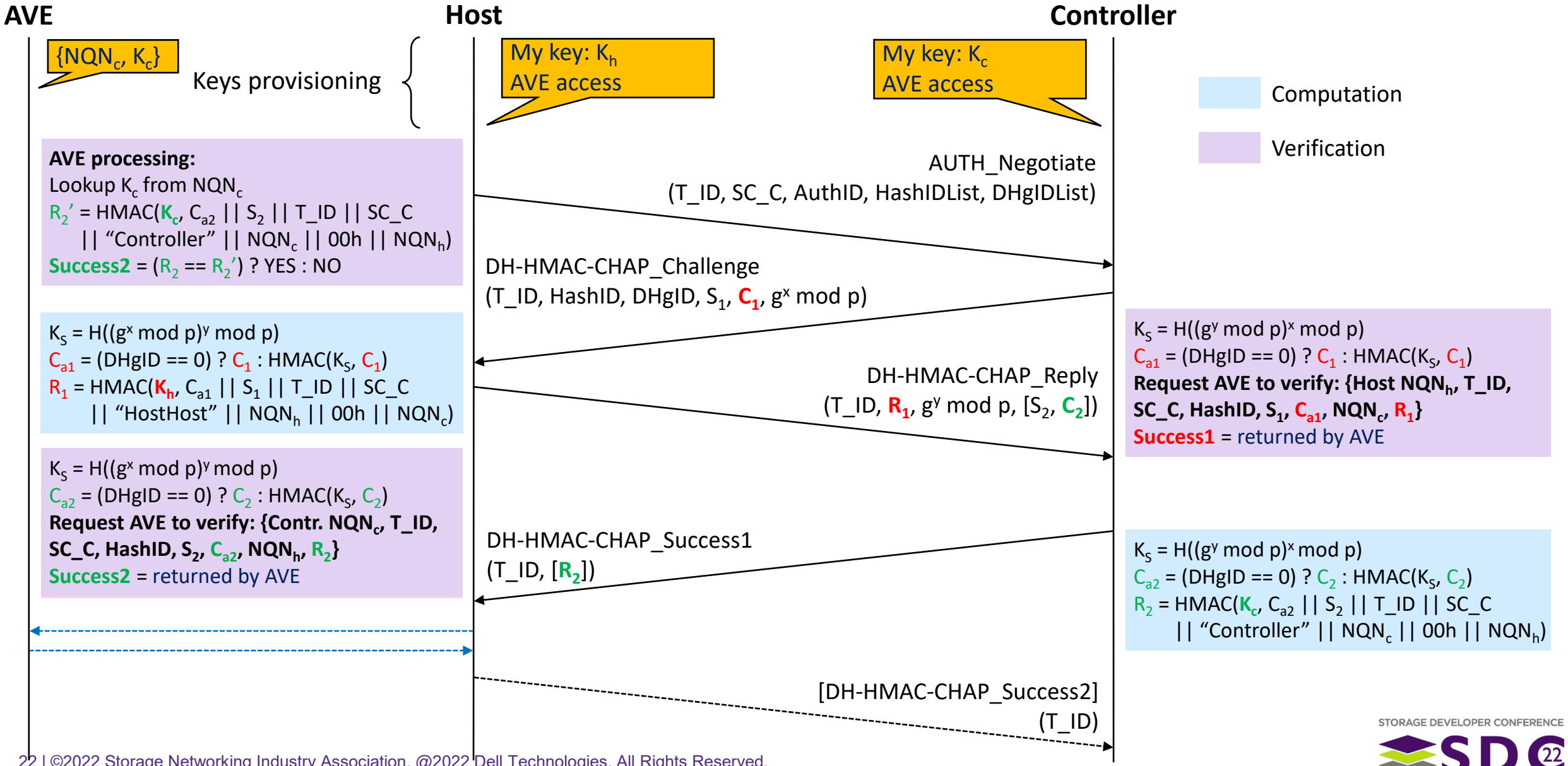
DH-HMAC-CHAP with AVE (1)



DH-HMAC-CHAP with AVE (2)

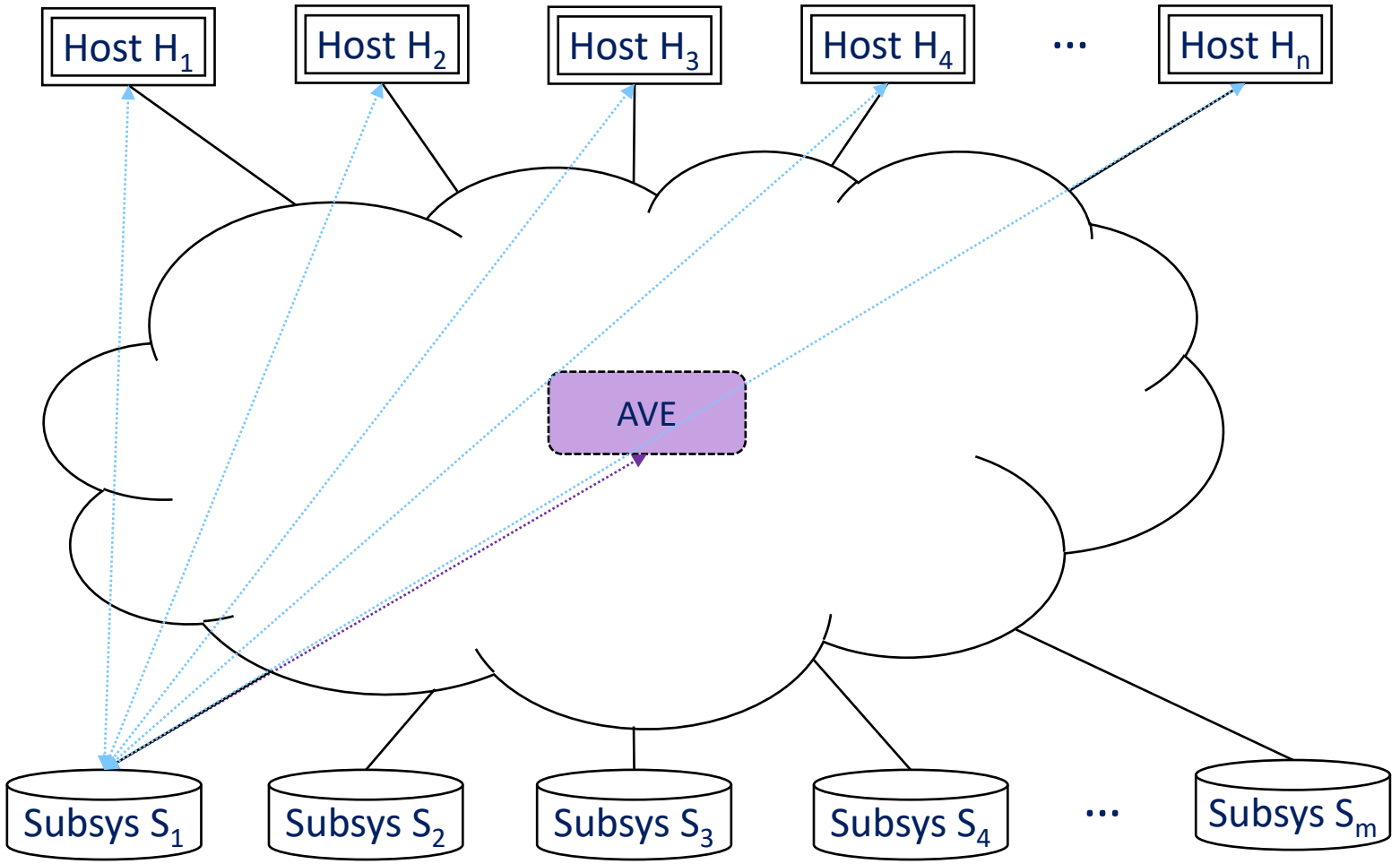


DH-HMAC-CHAP with AVE (3)



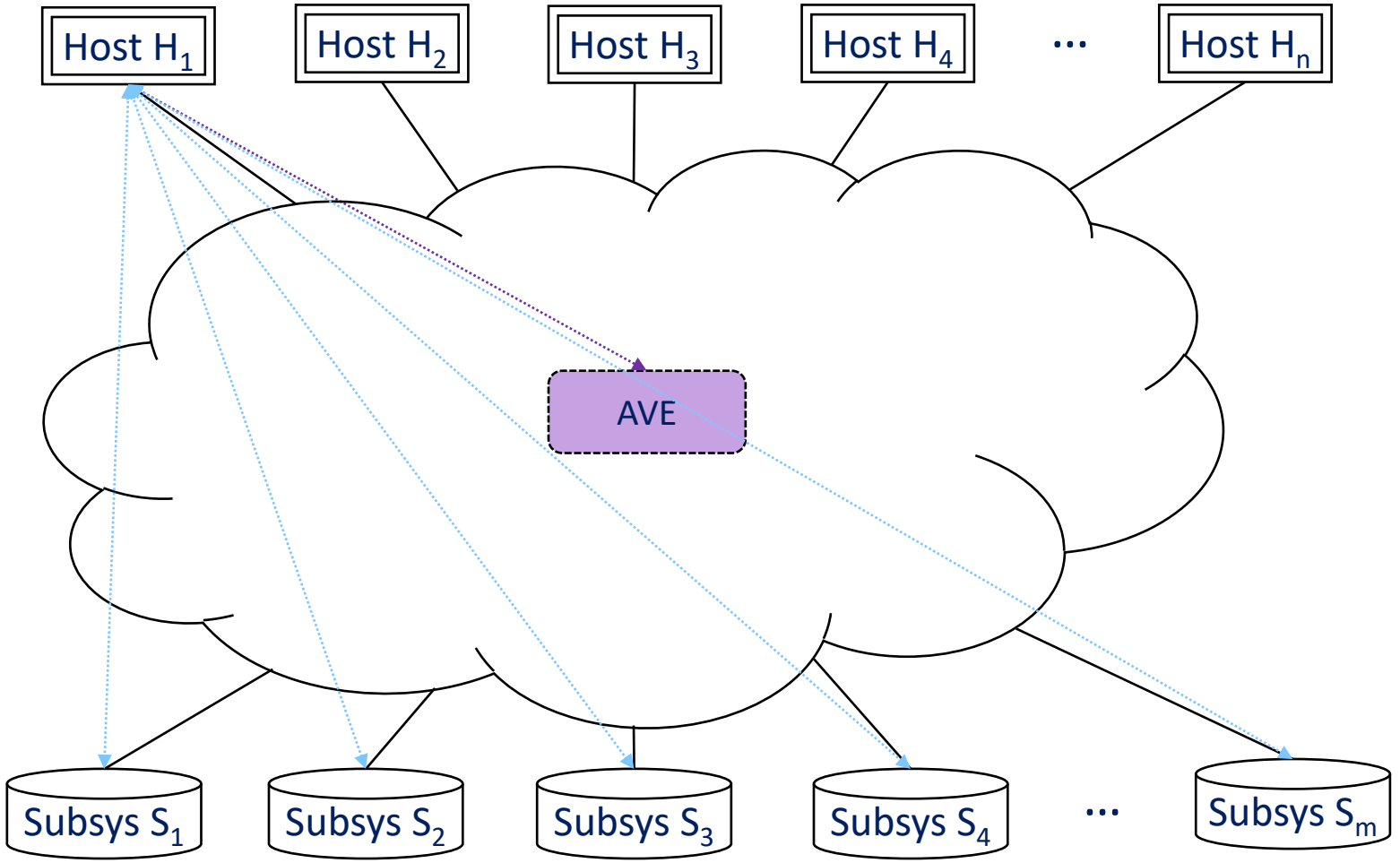
Subsystem Provisioning with AVE

- For subsystem S_1:
  - Computation key: K_{S1}
  - AVE access
- Subsystem provisioning completely independent from hosts provisioning
  - Subsystem configuration is simplified even if hosts do not use the AVE
- Similar configuration for other subsystems



Host Provisioning with AVE

- For host H_1:
  - Computation key: K_{H1}
  - AVE access
- Host provisioning completely independent from subsystems provisioning
  - Host configuration is simplified even if subsystems do not use AVE
- Similar configuration for other hosts



DH-HMAC-CHAP with AVE Provisioning Scaling Example

- Let's consider a fabric with 100 hosts and 25 subsystems
  - Suppose all hosts want to authenticate all subsystems
- Initial host provisioning:
  - Each host needs to be provisioned with its own key and the access to the AVE
- Subsystem addition to the fabric:
  - No need to touch any host, just add the additional subsystem key to the AVE
- Initial subsystem provisioning:
  - Each subsystem needs to be provisioned with its own key and the access to the AVE
- Host addition to the fabric:
  - No need to touch any subsystem, just add the additional host key to the AVE

Provisioning Scaling Comparison

- Let's consider a fabric with 100 hosts and 25 subsystems
  - Suppose all hosts want to authenticate all subsystems
- Initial host provisioning:
  - Plain DH-HMAC-CHAP: each host needs to be provisioned with its own key and the 25 subsystem verification keys
  - DH-HMAC-CHAP + AVE: each host needs to be provisioned with its own key and the access to the AVE
- Subsystem addition to the fabric:
  - Plain DH-HMAC-CHAP: all 100 hosts needs to be re-provisioned to add the additional subsystem verification key
  - DH-HMAC-CHAP + AVE: no need to touch any host, just add the additional subsystem key to the AVE
- Initial subsystem provisioning:
  - Plain DH-HMAC-CHAP: each subsystem needs to be provisioned with its own key and the 100 host verification keys
  - DH-HMAC-CHAP + AVE: each subsystem needs to be provisioned with its own key and the access to the AVE
- Host addition to the fabric:
  - Plain DH-HMAC-CHAP: all 25 subsystems needs to be re-provisioned to add the additional host verification key
  - DH-HMAC-CHAP + AVE: no need to touch any subsystem, just add the additional host key to the AVE
- In other words: An Authentication Verification Entity makes DH-HMAC-CHAP provisioning scalable

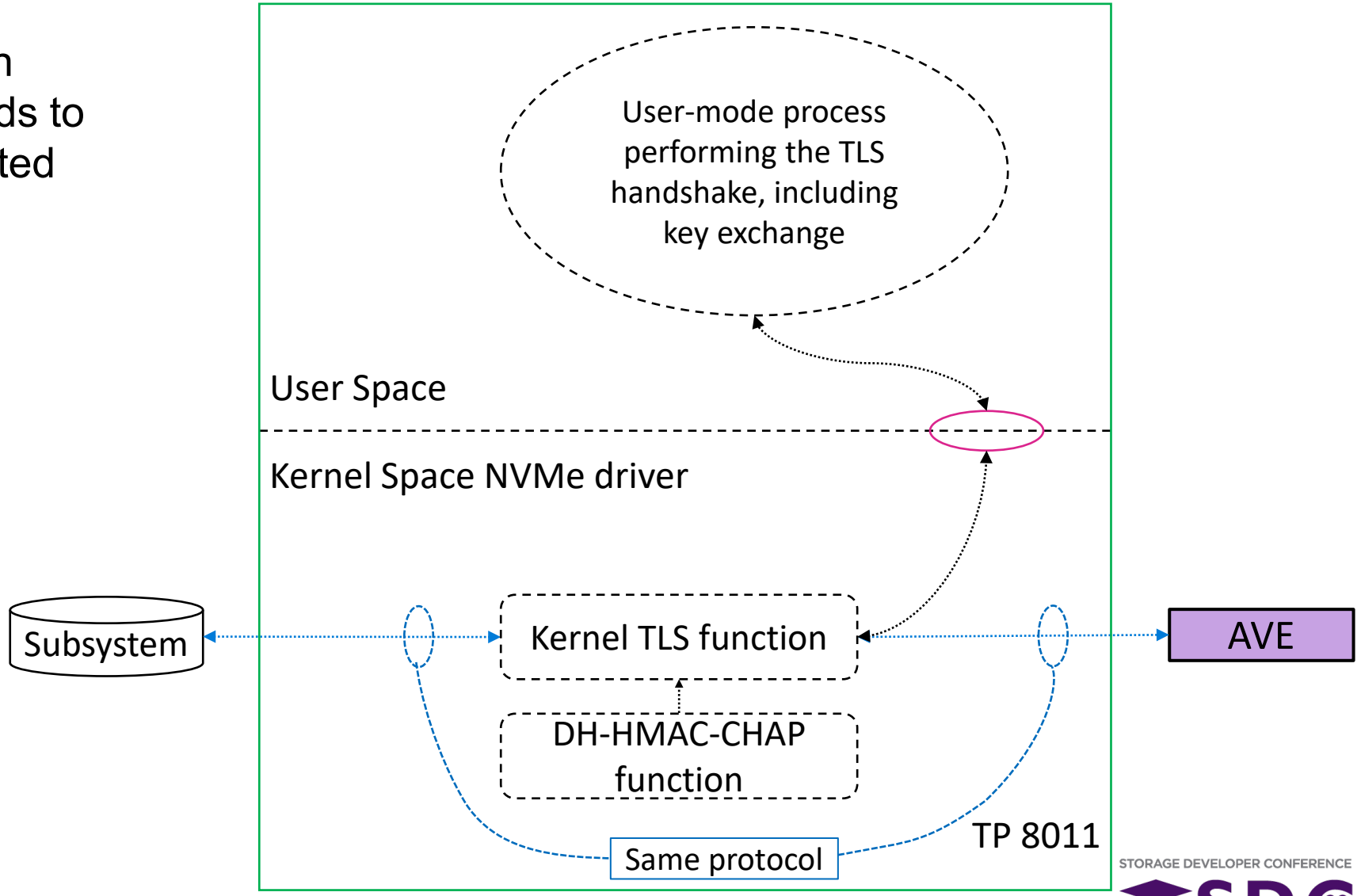
Ideal Fabric Authentication Provisioning

- Ideal characteristics:
  - Minimal initial provisioning
  - No need for endpoint provisioning updates when the fabric change
    - e.g., a subsystem is added or a host is added
  - Consistent behavior
    - i.e., independence from other entities
  - Automated
    - Manage security through a protocol rather than by requiring specialized skills
- Resulting deployment model:
  Provision authentication once and forget about it
- DH-HMAC-CHAP with AVE has these characteristics

AVE Access and DH-HMAC-CHAP Provisioning

AVE Access Protocol

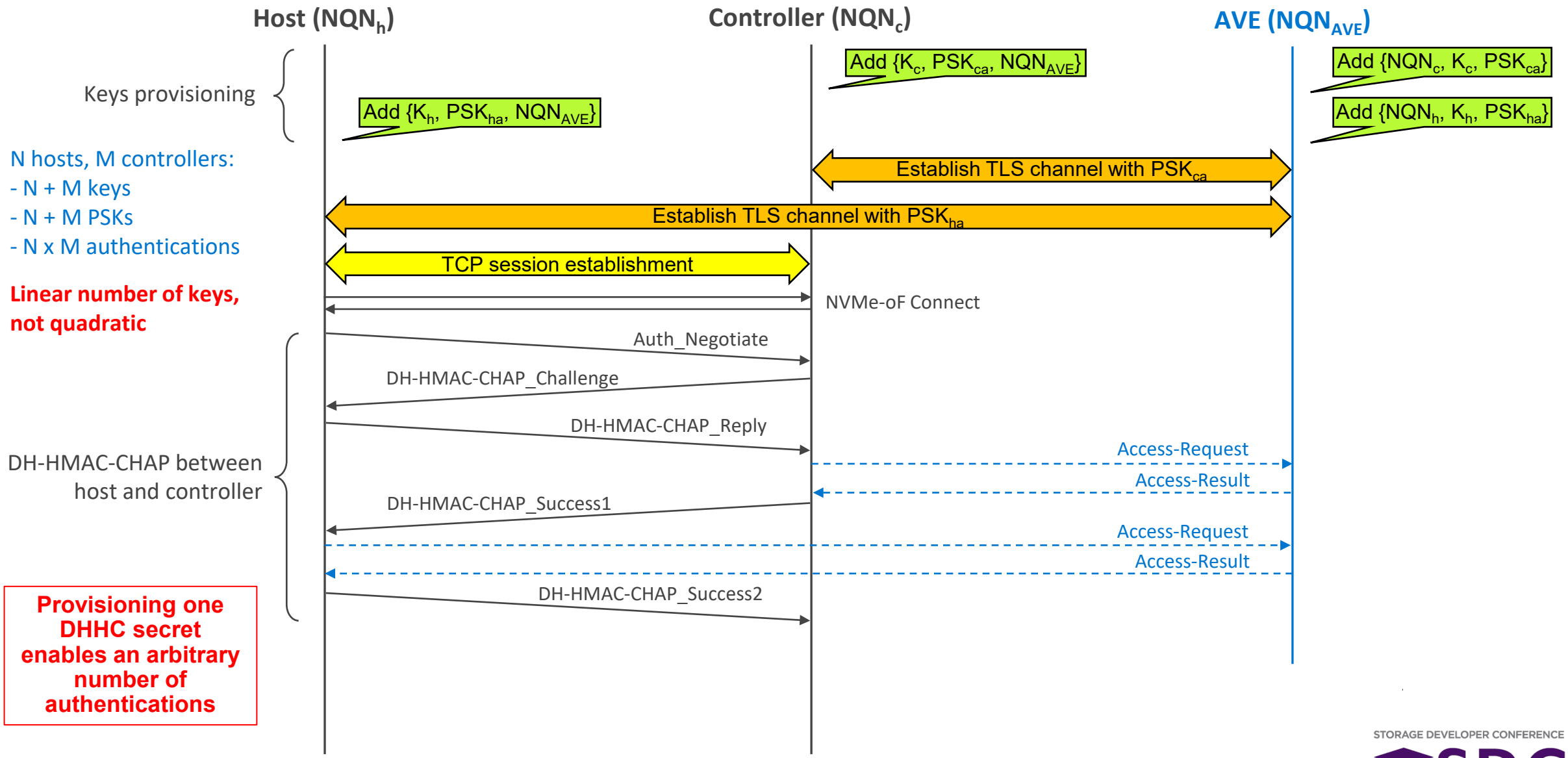
- The communication between host/controller and AVE needs to be authenticated and protected
  - Only authenticated and authorized entities shall be allowed to access the AVE
  - Integrity and confidentiality are required to protect the information exchanged
- Reuse the NVMe TLS specification
  - Enables kernel implementation
  - Identifies the AVE with an NQN (NQN_{AVE})



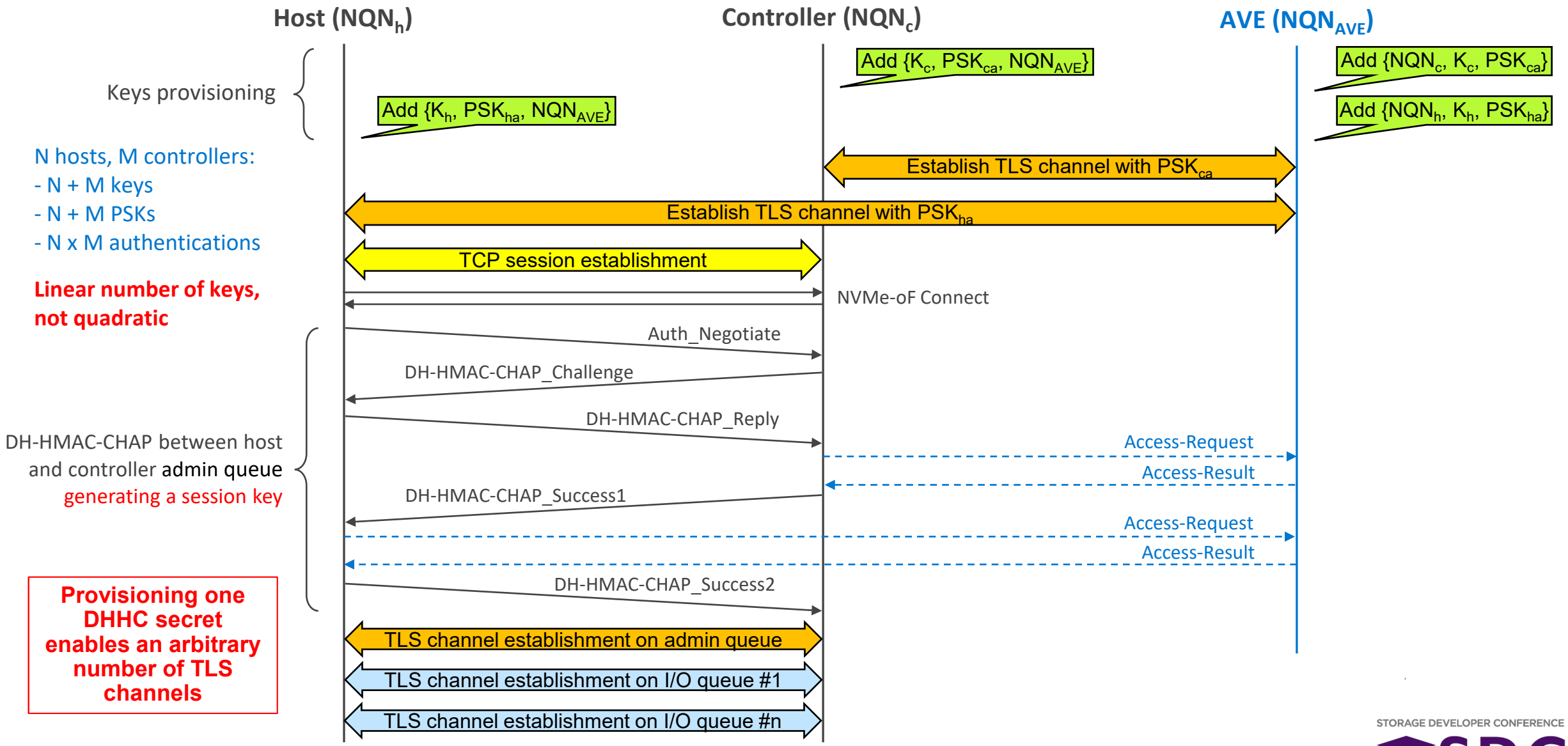
Provisioning of DH-HMAC-CHAP and AVE

- For an NVMe entity (host or controller) having NQN_e as NQN:
  - Generate a random DH-HMAC-CHAP key, K_e
  - Generate a random TLS PSK to connect with the AVE, PSK_{ea}
  - Add to the AVE database the record $\{NQN_e, K_e, PSK_{ea}\}$
  - Provision $\{K_e, PSK_{ea}, NQN_{AVE}\}$ to the NVMe-oF entity

Eample of DH-HMAC-CHAP with AVE



Concatenating TLS to DH-HMAC-CHAP with AVE



Summary

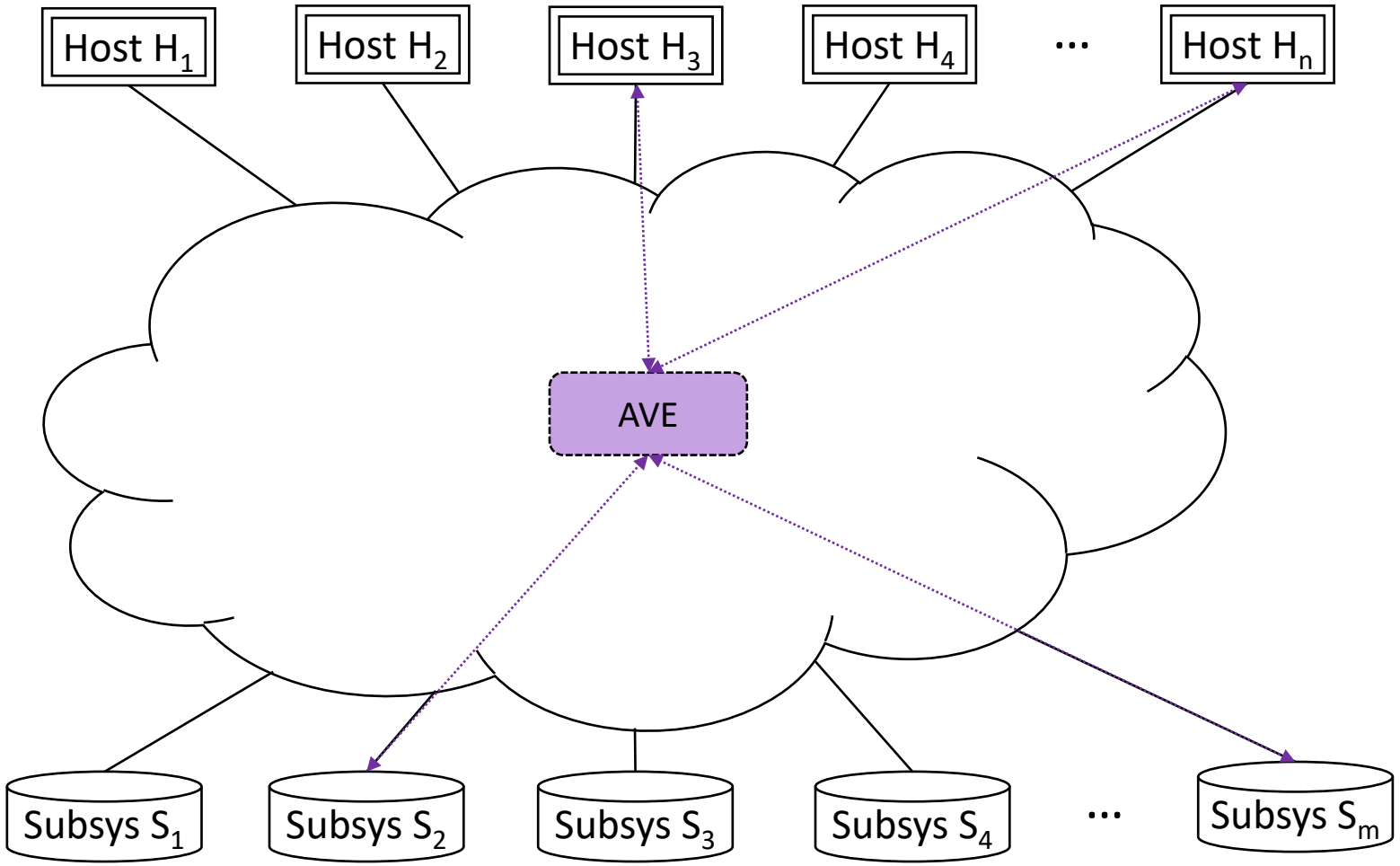
- Provisioning of the current NVMe security protocols does not scale
  - TLS requires provisioning one PSK per each pair of entity of an NVMe fabric
  - Plain DH-HMAC-CHAP requires provisioning verification keys
- Adding the DH-HMAC-CHAP Authentication Verification Entity (AVE) to the NVMe architecture makes provisioning those protocol scalable
  - Enables “provision authentication once and forget about it”
  - It is being worked out in TP 8019

Please take a moment to rate this session.

Your feedback is important to us.

Heterogeneous Configuration: Host Provisioning

- Host H_1: No AVE
  - Computation key: K_{H1}
  - Subsystem verification keys:
    - $\{NQN_{S1}, K_{S1}\}$
    - $\{NQN_{S2}, K_{S2}\}$
    - $\{NQN_{S3}, K_{S3}\}$
    - $\{NQN_{S4}, K_{S4}\}$
    - …
    - $\{NQN_{Sm}, K_{Sm}\}$
- Same for hosts not using AVE
- Host H_n: AVE
  - Computation key: K_{Hn}
  - AVE access
- Same for hosts using AVE



Heterogeneous Configuration: Subsystem Provisioning

- Subsystem S_1: no AVE
  - Computation key: K_{S1}
  - Host verification keys:
    - $\{NQN_{H1}, K_{H1}\}$
    - $\{NQN_{H2}, K_{H2}\}$
    - $\{NQN_{H3}, K_{H3}\}$
    - $\{NQN_{H4}, K_{H4}\}$
    - …
    - $\{NQN_{Hn}, K_{Hn}\}$
- Same for subsystems not using AVE
- Subsystem S_m: AVE
  - Computation key: K_{Sm}
  - AVE access
- Same for subsystems using AVE