STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021
A SNIA Event

Securing an NVMe-oF IP Fabric

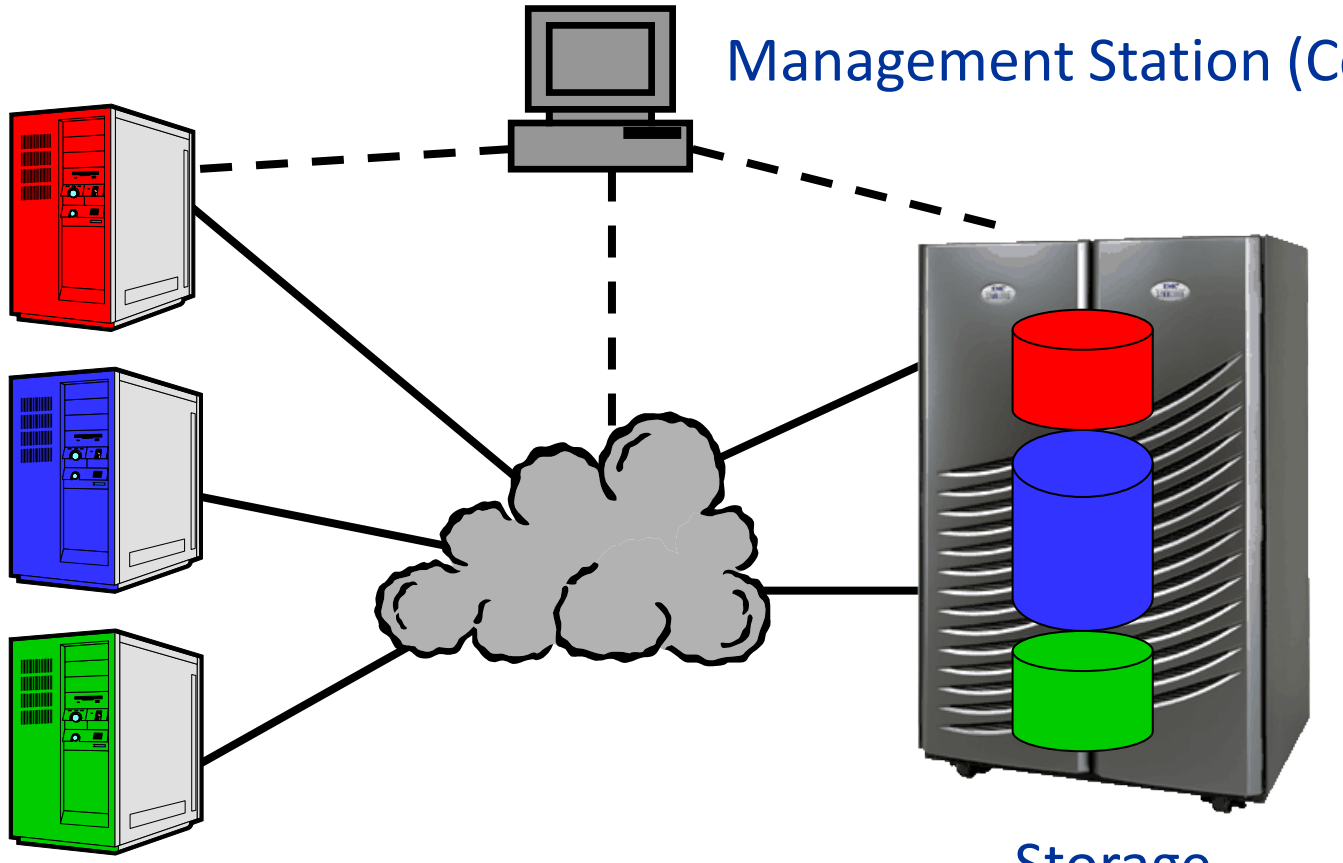
Claudio DeSanti
Distinguished Engineer
Dell Technologies CTIO Group

Agenda

- SAN Security Framework
- NVMe-oF Security: Authentication
- NVMe-oF Security: Secure Channel

SAN Security Framework

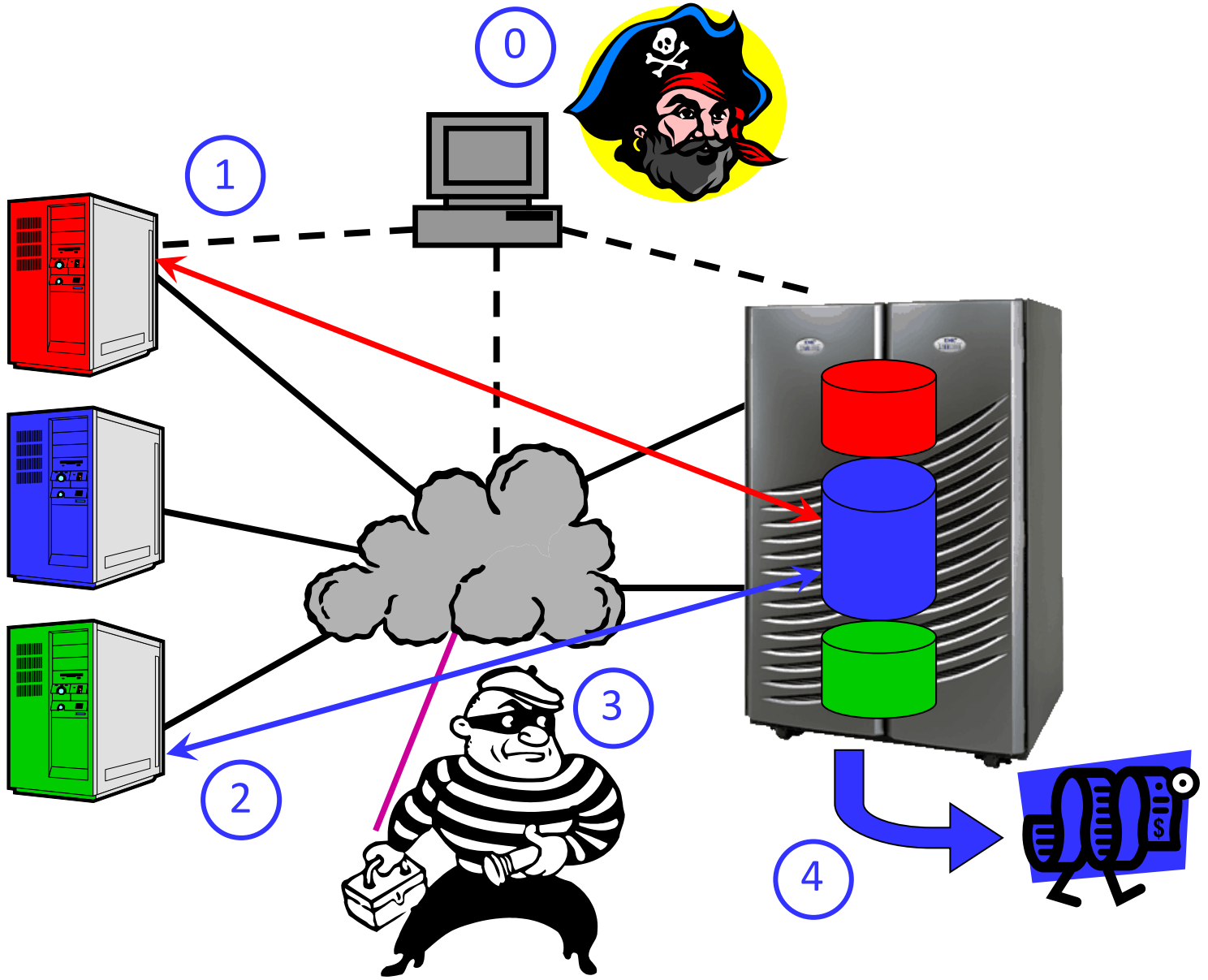
Storage Area Network (SAN) Example



Images credit: David Black

Security Threats

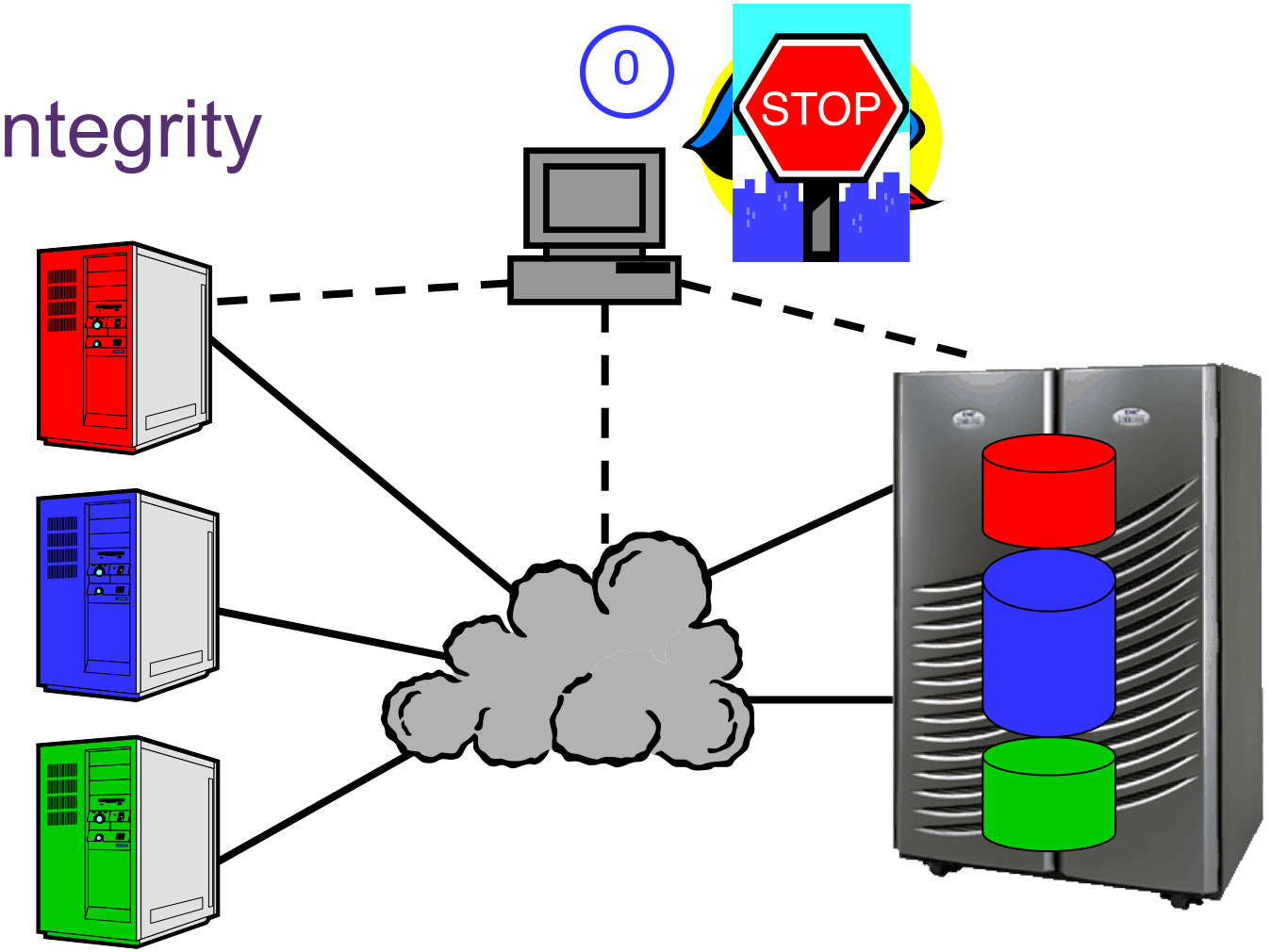
0) Management & System Integrity
1) Uncontrolled Storage Access
2) Impersonation (Spoofing)
3) Communication Access
   - Eavesdrop
   - Inject/Modify
4) External Access
   - Media Theft
   - Other Access



Security Threat 0: Management & System Integrity

0) Management & System Integrity

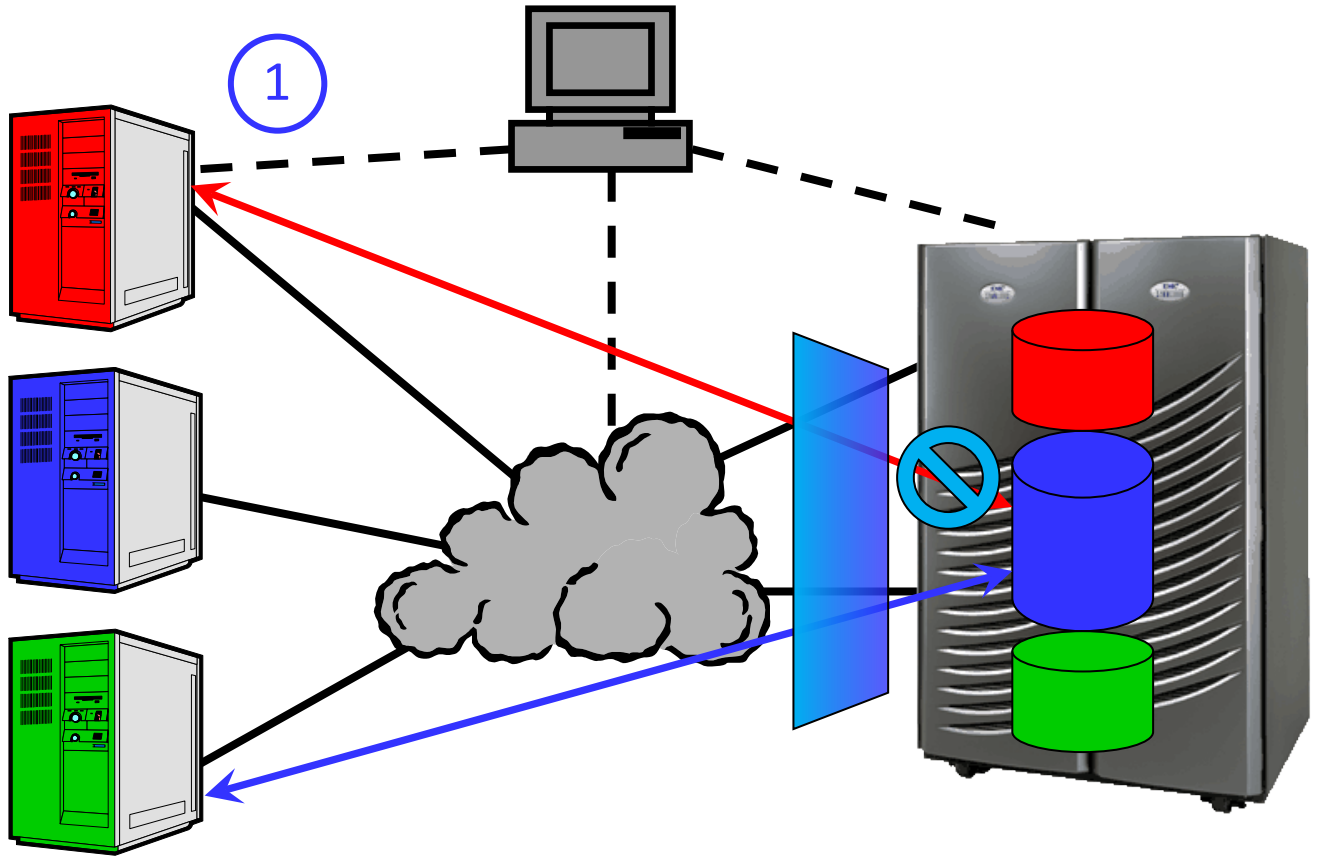
- Countermeasures: Management Security
  - Authentication & Authorization
  - Logging and Anomaly Detection
  - Secure Channels
- Countermeasures: System Integrity
  - Hardware/software/firmware integrity checks and assurance
  - Preferably anchored to hardware root of trust



Security Threat 1: Access Control

1) Uncontrolled Storage Access

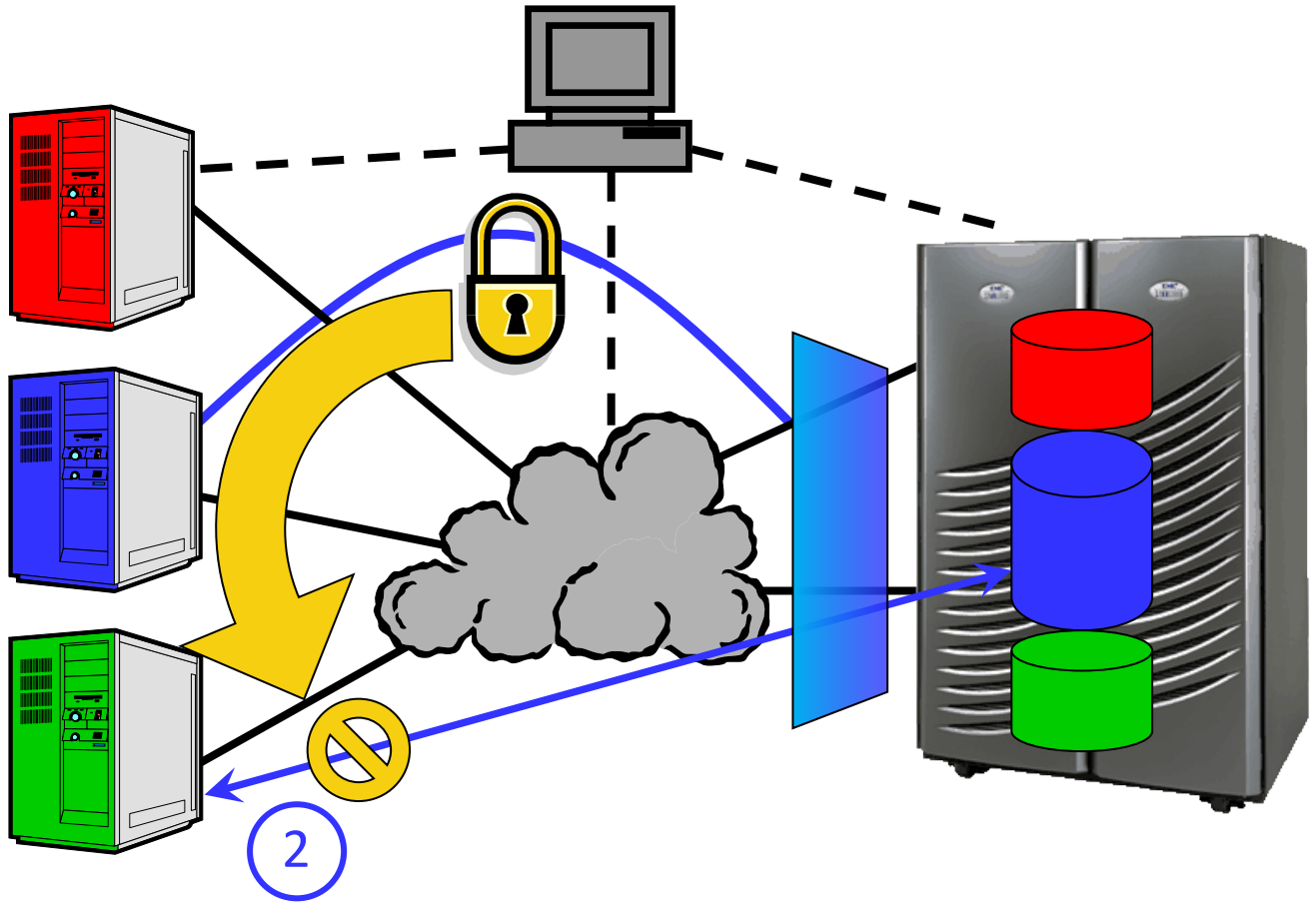
- Countermeasure: Storage Access Control
  - E.g., FC zoning, SCSI LUN masking, NVMe Namespace mapping
- Does not prevent impersonation



Security Threat 2: Impersonation

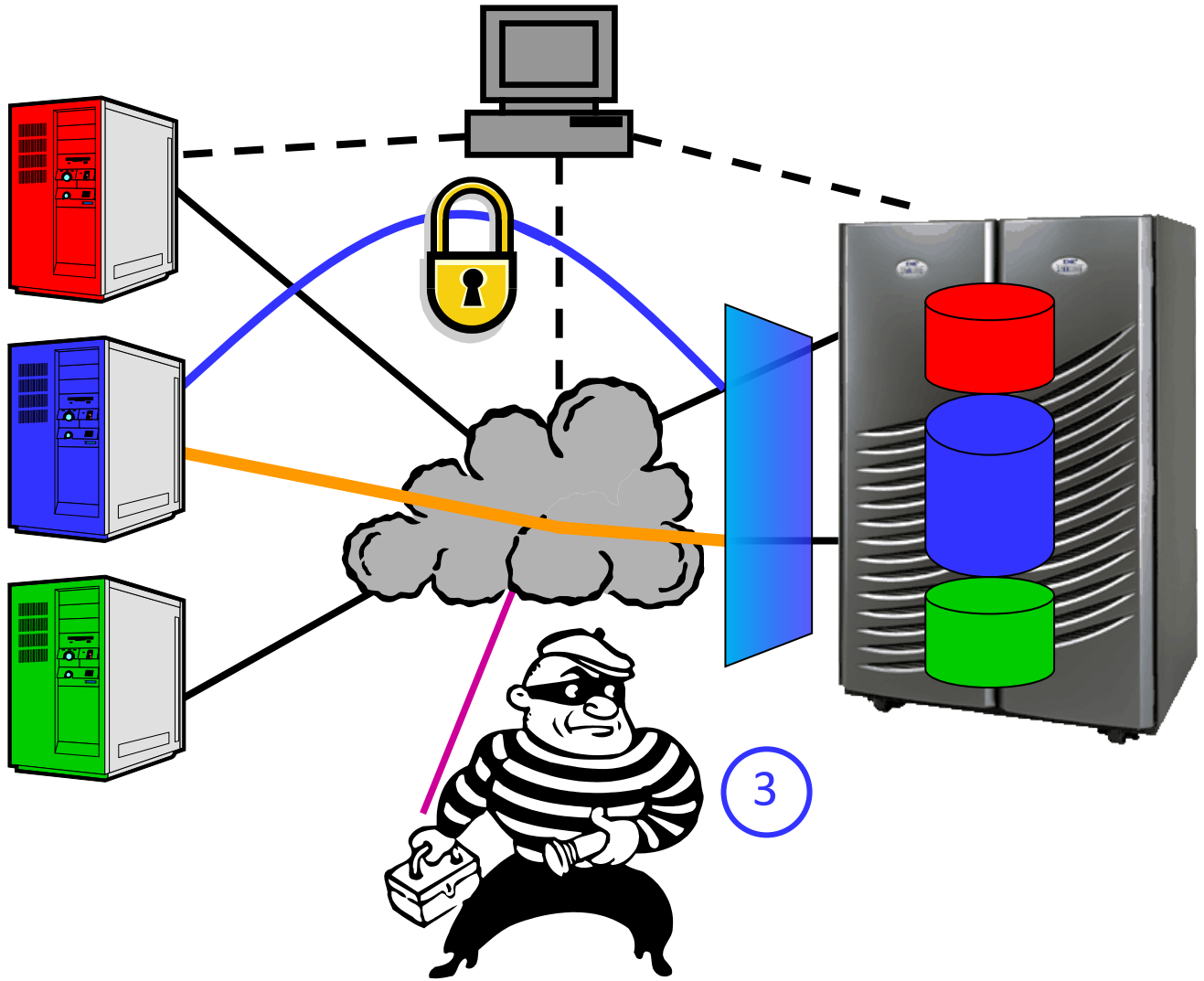
2) Impersonation (Spoofing)

- Countermeasure: Authentication
  - Proof of identity



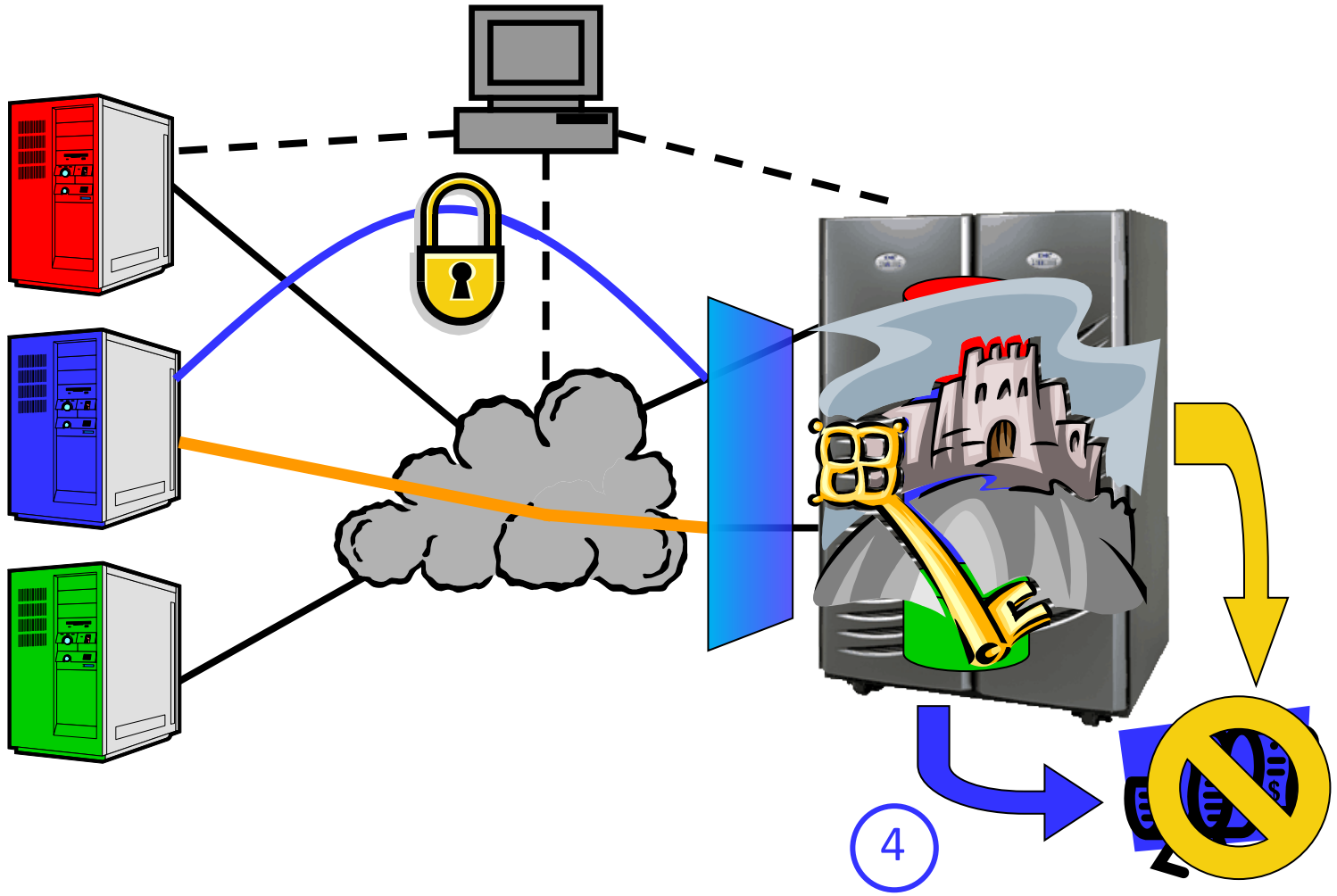
Security Threat 3: Communication

3) Communication Access
   - Eavesdrop
   - Inject/Modify
- Countermeasure: Secure Channel (data in flight)
   - Confidentiality
   - Cryptographic Integrity



Security Threat 4: Stored Data

4) External Access
   - Media Theft
   - Other Access
- Countermeasure: Stored Data Encryption (data at rest)
   - Application, server OS, VM guest OS
   - Hypervisor
   - Storage drives (SEDs)



Storage Networking Security Review

0) Management & System Integrity

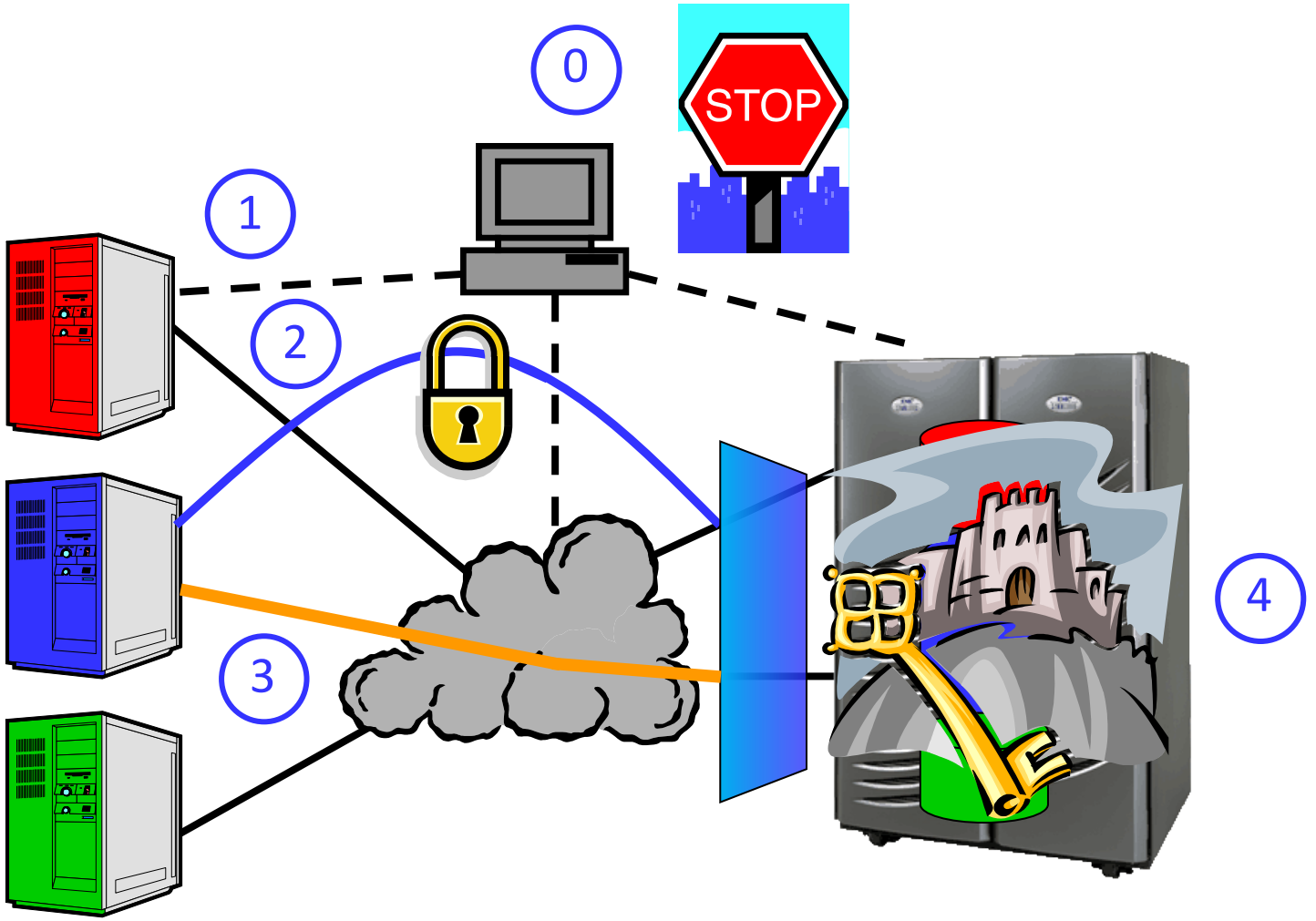
1) Storage Access Control

2) Authentication (proof of identity)

3) Secure Channel (data in flight)
   - Confidentiality
   - Cryptographic Integrity

4) Stored Data (data at rest) Encryption

Focus of this Discussion

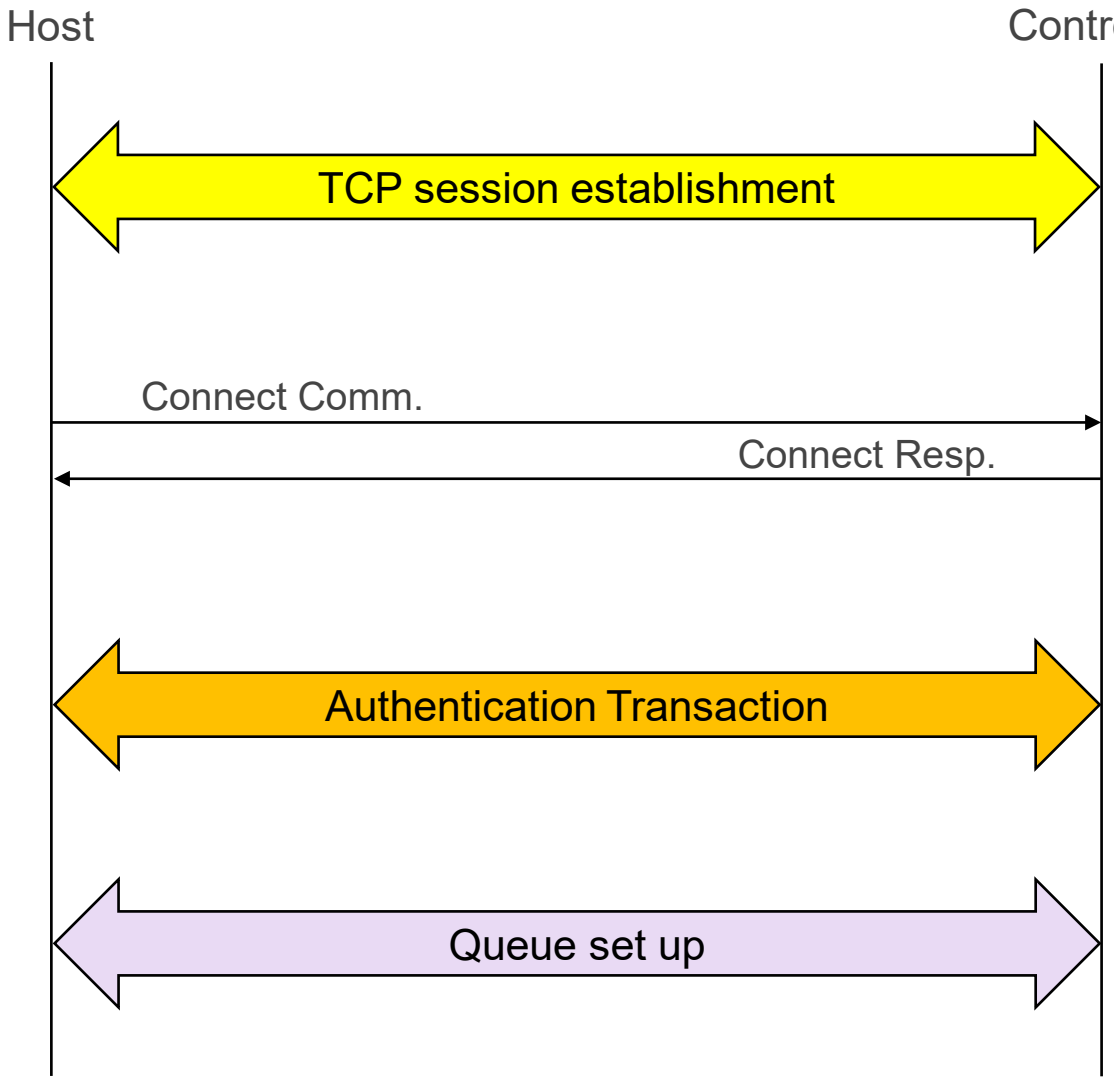


NVMe-oF Security: Authentication

SAN Protocols - Security Mechanism Comparison

	iSCSI	Fibre Channel	NVMe over Fabrics/IP
Storage Endpoint Authentication	**CHAP** (strong secret) **SRP** (weak secret, e.g., password) [not used in practice]	**DH-CHAP** (strong secret) **FCPAP** (weak secret, e.g., password) **FCAP** (certificates) **FC-EAP** (strong secret)	**DH-HMAC-CHAP** (strong secret)
Secure Channel (authenticated encryption & cryptographic integrity)	**IPsec** (e.g., in security gateway)	**FC ESP_Header** (IPsec-like, limited usage)	**TLS** (pre-shared key) – for TCP only **IPsec** is also an option

NVMe-oF Authentication Example

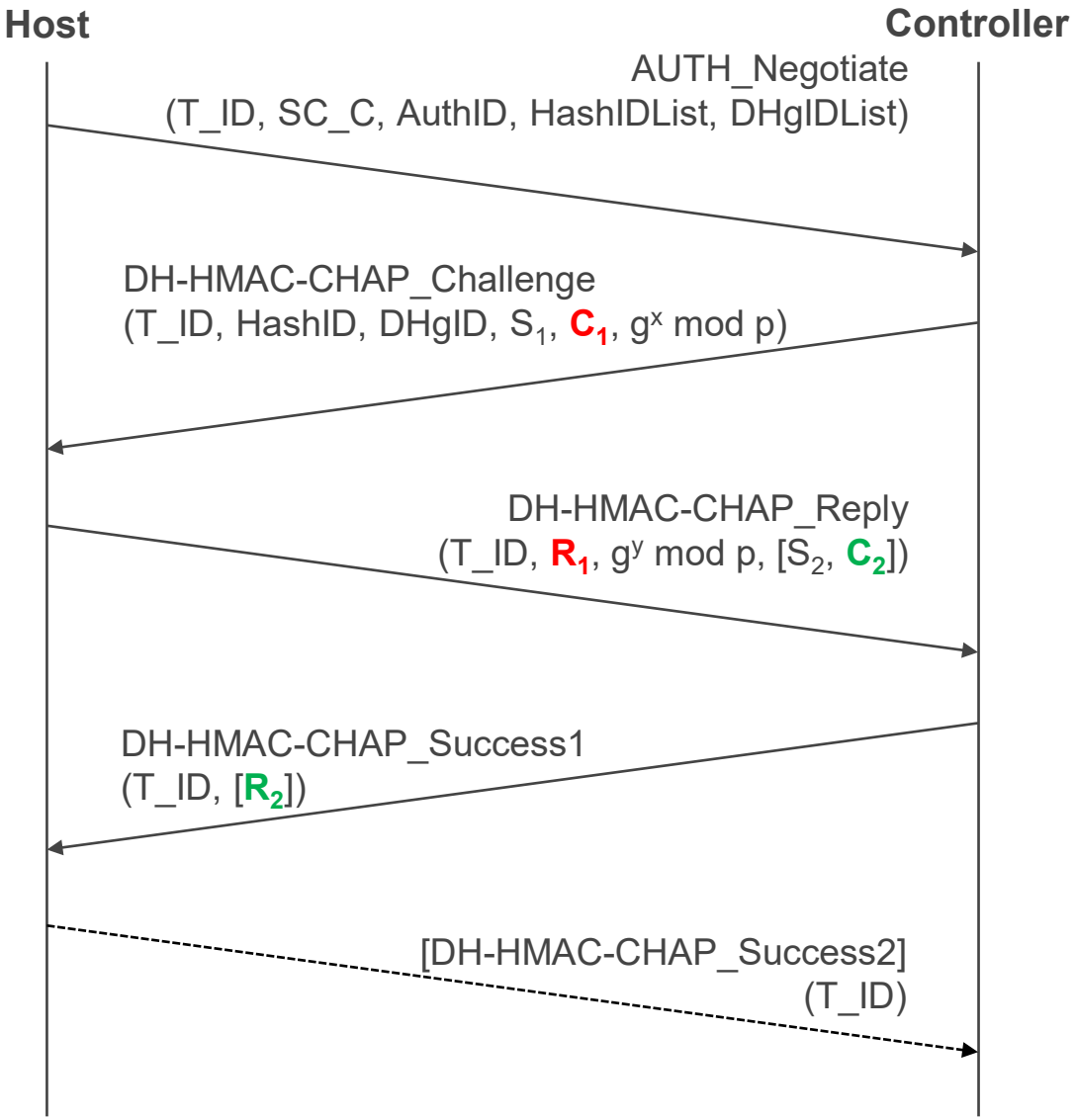


1. A TCP session is established
2. The Connect exchange is performed to set up NVMe queue and associate host to controller
3. The host performs an authentication transaction with the controller to authenticate the end-points
4. Queue is ready for subsequent operations

NVMe-oF Authentication: DH-HMAC-CHAP protocol

- Defined in NVMe 2.0 base specification (formerly in TP 8006)
- Based on keys that need to be different for each NQN
- Challenge/response protocol: CHAP
  - the authenticator sends a challenge C;
  - the responder computes a response R = Hash(C || $key_{responder}$ || other things);
  - the authenticator verifies the response (or delegates verification)
- DH-HMAC-CHAP: Strengthened version of CHAP
  - DH: Diffie-Hellman, adds (optional) key exchange to frustrate eavesdroppers
  - HMAC: Hashed MAC, uses secure hash twice to improve security
- Bidirectional authentication

DH-HMAC-CHAP Protocol: Bidirectional Authentication



Unidirectional challenge/response protocol

- Controller C sends a challenge C_1
- Host H computes a response R_1 = HMAC(K_h, C_1 || other things)
- Controller C verifies the response
- Unidirectional authentication (controller authenticates host)

Getting bidirectional authentication

- H sends a challenge C_2
- C computes a response R_2 = HMAC(K_c, C_2 || other things)
- H verifies the response
- Unidirectional authentication (host authenticates controller)

Verification:

- Controller computes R_1' and check if it matches the received R_1
- Host computes R_2' and check if it matches the received R_2

Authentication Responses

$K_S = H((g^x \bmod p)^y \bmod p)$
$C_{a1} = (DHgID == 0)\ ?\ C_1 : HMAC(K_S, C_1)$
$R_1 = HMAC(K_h, C_{a1} \| S_1 \| T\_ID \| SC\_C \| \text{“HostHost”} \| NQN_h \| 00h \| NQN_c)$

$K_S = H((g^y \bmod p)^x \bmod p)$
$C_{a2} = (DHgID == 0)\ ?\ C_2 : HMAC(K_S, C_2)$
$R_2 = HMAC(K_c, C_{a2} \| S_2 \| T\_ID \| SC\_C \| \text{“Controller”} \| NQN_c \| 00h \| NQN_h)$

DH-HMAC-CHAP Secret Provisioning

- ‘Secrets’ are provisioned and transformed into ‘keys’, used by the protocol
- Standardized secret interchange ASCII format:
  - DHHC-1:xx:<Base64 encoded string>:
    - ‘xx’ identifies the hash function to use to transform a secret in a key
    - Key = HMAC(secret, NQN || ”NVMe-over-Fabrics”)
- An NVMe-oF entity:
  - For authentication needs to be provisioned with its own secret
  - For verification needs to know the key of the other entity
- Centralized authentication verification drastically simplifies the provisioning process
  - Only the entity’s secret needs to be provisioned, no other keys

NVMe-oF Security: Secure Channel

SAN Protocols - Security Mechanism Comparison

	iSCSI	Fibre Channel	NVMe over Fabrics/IP
Storage Endpoint Authentication	**CHAP** (strong secret) **SRP** (weak secret, e.g., password) [not used in practice]	**DH-CHAP** (strong secret) **FCPAP** (weak secret, e.g., password) **FCAP** (certificates) **FC-EAP** (strong secret)	**DH-HMAC-CHAP** (strong secret)
Secure Channel (authenticated encryption & cryptographic integrity)	**IPsec** (e.g., in security gateway)	**FC ESP_Header** (IPsec-like, limited usage)	**TLS** (pre-shared key) – for TCP only **IPsec** is also an option

Secure Channel: TLS

- TLS (Transport Layer Security): Widely used secure channel protocol
  - Secure channel = authentication, confidentiality, cryptographic integrity (primary properties)
  - Typical (web) usage: Server uses certificate with TLS, client authenticates after TLS setup (e.g., TLS-protected HTTP)
- TLS versions:
  - TLS 1.0 and 1.1: Obsolete - should not be used
  - TLS 1.2: Baseline TLS version, widely implemented and used, getting replaced by TLS 1.3
  - TLS 1.3: New version, complete protocol redesign (TLS 2.0 in practice), usage rolling out
    - Support available in some security libraries (e.g. OpenSSL & LibreSSL), expanding to more
- TLS 1.3 for NVMe/TCP: part of NVMe TCP transport specification (formerly in TP 8011)
  - TLS not specified for other NVMe-oF IP-based protocol (i.e., RDMA, e.g., RoCEv2)
  - Based on pre-shared keys (PSKs)
- NVMe/TCP TLS 1.2: discouraged by TP 8011
  - Usage was specified by NVMe-oF 1.1 standard
  - Implementation limitations on maximum identity size make problematic its support

Why TLS 1.3?

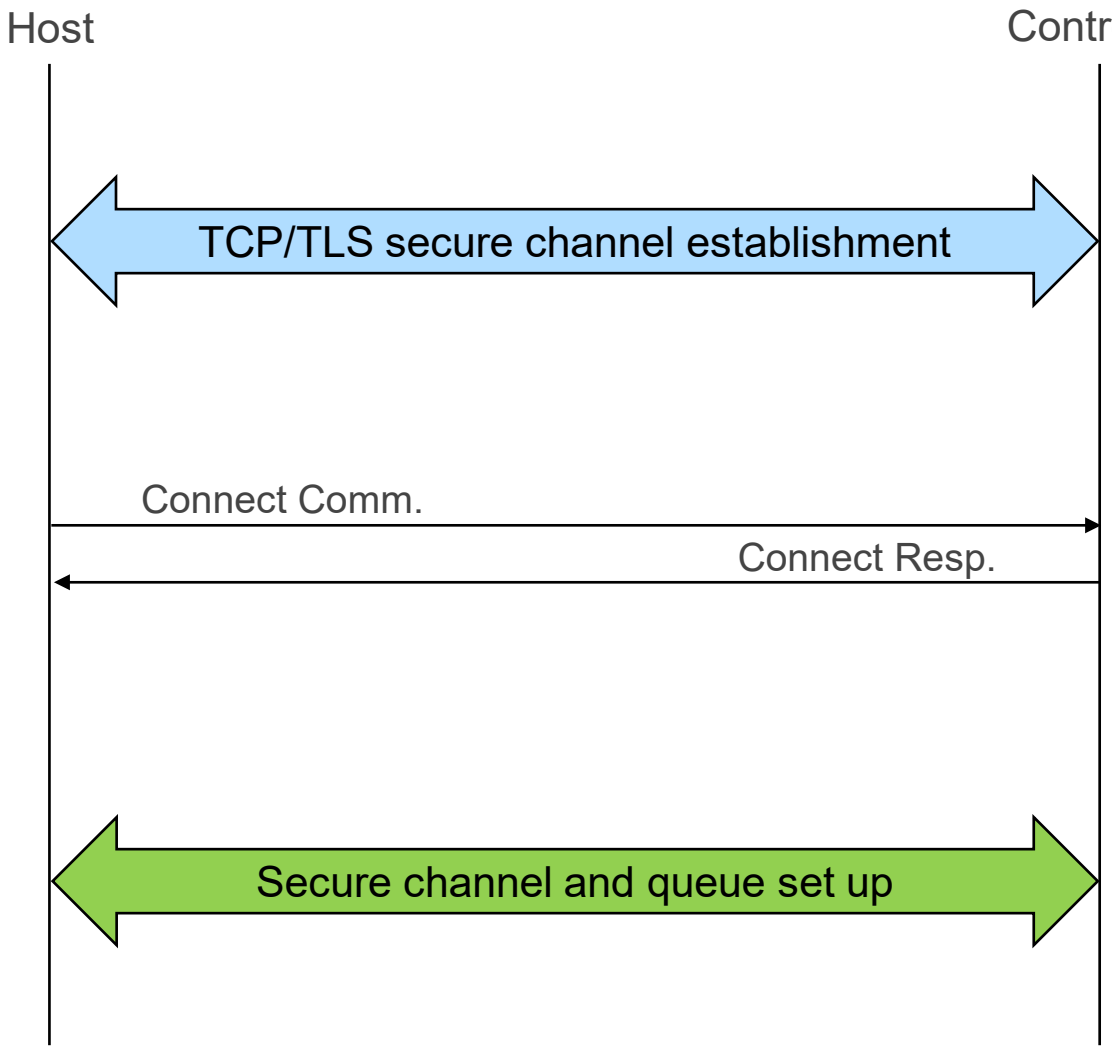
TLS 1.2

- IANA TLS registry has 300+ cipher suite code points
  - Uncertain security properties, difficult interoperability
- Encryption starts late in the handshake
  - Client cert and target site are sent in the clear
  - Poor privacy
- Many features with known security flaws

TLS 1.3

- 5 cipher suites, all with PFS and modern algorithms
  - Consistent security properties
- Encryption starts as early as possible, hiding content length
  - Minimal set of cleartext protocol bits on the wire
  - Less user information visible to the network
- All those features are omitted from 1.3

TLS 1.3



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

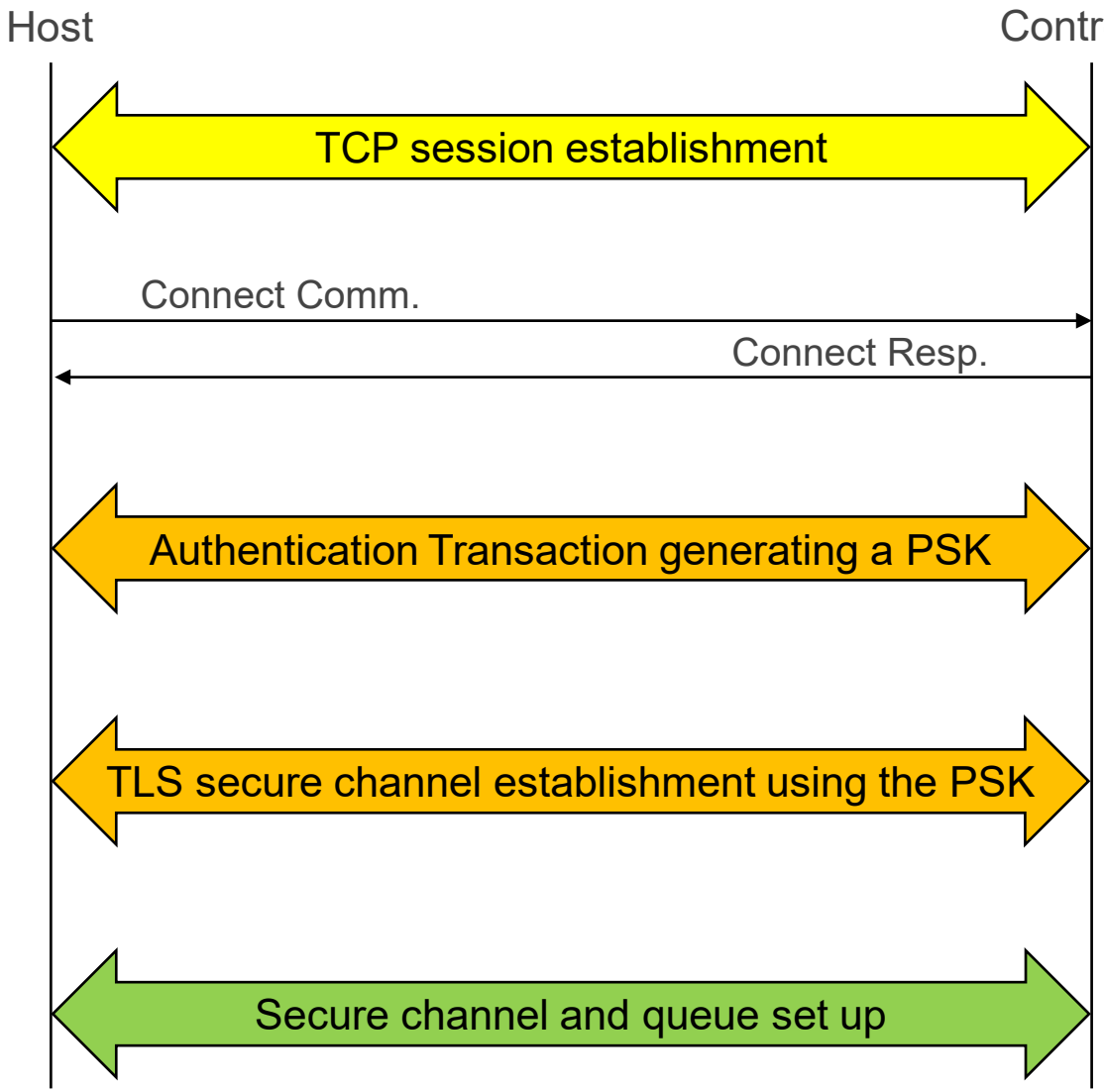
2. The Connect exchange is performed to set up NVMe queue and associate host to controller

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 be configured with the same PSK
  - This can lead to a deployment option called ‘group PSK’: all NVMe entities share the same PSK
  - Big security concern (compromising a single node may allow an attacker to access all secure channels)
  - The proper way would be to have a PSK per each pair of entity that can communicate (n^2 problem)
- Authentication protocols to the rescue
  - Upon successful completion of an authentication exchange, the two involved NVMe entities generate an ephemeral shared session key (e.g., a ‘PSK’ computed on the fly)
  - The TLS negotiation can then be performed using a PSK derived from that shared key
    - No more need for ‘group PSK’
  - Implementation result: the TCP connection begins unsecured and then transitions to secured
    - Opportunistic TLS
  - Linear problem (not anymore n^2): need just one secret per entity

Authentication Followed by TLS



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

Keys Derivation

$$K_S = H((g^x \bmod p)^y \bmod p) = H((g^y \bmod p)^x \bmod p) = H(g^{xy} \bmod p)$$

$C_{a1} = HMAC(K_S, C_1)$

$C_{a2} = HMAC(K_S, C_2)$

Generated PSK for TLS = $HMAC(K_S, C_1 \,||\, C_2)$

$R_1 = HMAC(K_h, C_{a1} \,||\, S_1 \,||\, T\_ID \,||\, SC\_C \,||\, \text{“HostHost”} \,||\, NQN_h \,||\, 00h \,||\, NQN_c)$

$R_2 = HMAC(K_c, C_{a2} \,||\, S_2 \,||\, T\_ID \,||\, SC\_C \,||\, \text{“Controller”} \,||\, NQN_c \,||\, 00h \,||\, NQN_h)$

Takeaways

- The NVMe 2.0 set of specifications include a comprehensive security architecture
- The architecture defines protocols to address the fundamental SAN Security threats
  - Authentication (proof of identity)
  - Secure channel (data in flight)
- Authentication protocol: DH-HMAC-CHAP
- Secure Channel protocol: TLS 1.3

Please take a moment to rate this session

Your feedback is important to us

