LEGAL NOTICE:

This NVM Express over Fabrics revision 1.1 technical proposal is proprietary to the NVM Express, Inc. (also referred to as "Company") and/or its successors and assigns.

NOTICE TO USERS WHO ARE NVM EXPRESS, INC. MEMBERS: Members of NVM Express, Inc. have the right to use and implement this NVM Express over Fabrics revision 1.1 technical proposal subject, however, to the Member's continued compliance with the Company's Intellectual Property Policy and Bylaws and the Member's Participation Agreement.

NOTICE TO NON-MEMBERS OF NVM EXPRESS, INC.: If you are not a Member of NVM Express, Inc. and you have obtained a copy of this document, you only have a right to review this document or make reference to or cite this document. Any such references or citations to this document must acknowledge NVM Express, Inc. copyright ownership of this document. The proper copyright citation or reference is as follows: **"© 2007 - 2021 NVM Express, Inc. ALL RIGHTS RESERVED."** When making any such citations or references to this document you are not permitted to revise, alter, modify, make any derivatives of, or otherwise amend the referenced portion of this document in any way without the prior express written permission of NVM Express, Inc. Nothing contained in this document shall be deemed as granting you any kind of license to implement or use this document or the specification described therein, or any of its contents, either expressly or impliedly, or to any intellectual property owned or controlled by NVM Express, Inc., including, without limitation, any trademarks of NVM Express, Inc.

LEGAL DISCLAIMER:

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "**AS IS**" BASIS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, NVM EXPRESS, INC. (ALONG WITH THE CONTRIBUTORS TO THIS DOCUMENT) HEREBY DISCLAIM ALL REPRESENTATIONS, WARRANTIES AND/OR COVENANTS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT COMMON LAW, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, VALIDITY, AND/OR NONINFRINGEMENT.

All product names, trademarks, registered trademarks, and/or servicemarks may be claimed as the property of their respective owners.

NVM Express Workgroup
c/o VTM Group
3855 SW 153rd Drive
Beaverton, OR 97003 USA
Info@nvmexpress.org

NVM Express Technical Proposal for New Feature

Technical Proposal ID	**8006 - NVMe-oF In Band Authentication**
Change Date	**03/17/2021**
Builds on Specification	**NVM Express over Fabrics 1.1, TP 8011**

Technical Proposal Author(s)

Name	Company
Vladislav Bolkhovitin, Yoni Shternhell, Christoph Hellwig	WDC
David Black, Claudio DeSanti	Dell EMC
Costa Sapuntzakis	Pure Storage

This technical proposal defines the Diffie-Hellman HMAC-CHAP (DH-HMAC-CHAP) in-band authentication protocol for NVMe over Fabrics. The authors would like to acknowledge Charlie Kaufman for his contributions.

Revision History	
Revision Date	**Change Description**
4/30/2018	Final internal draft
5/25/2018	Updated based of David Black comments
6/01/2018	Updates and Authentication Address Mix added
6/13/2018	N-SRP added
6/18/2018	N-SRP updates
7/11/2018	Minor clarifications and updates during implementation
7/18/2018	Use single ID for SECP field and minor clarifications and fixes
7/19/2018	Use values of 1 and 2 for SPSP0 field and minor clarifications and fixes
7/25/2018	Clarifications about DH exponent sizes and DH parameter validation added
7/27/2018	Clarifications about authentication elements sizes added, notes about DH parameters reuse and parallel connects added, NSRP simplifications, change to HMAC function to generate N-SRP verifiers, minor cleanups
8/27/2018	N-SRP: internal hash introduced, editorial modifications
08/30/2018	Whirlpool, SM3 and MD5 hashes removed
09/24/2018	Clean version to start Phase 3 discussions
10/04/2018	Modification upon comments received at Oct 4 call
10/18/2018	Add reference to One-Way Hash and Internal Hash tables
11/26/2018	- Modifications to reflect Intel's comments from 11/15 - HMAC() usage changed for DH-HMAC-CHAP - Address Mix clarifications and minor cleanups
12/13/2018	Address Mix removed
1/28/2019	Serious rewrite of common material and DH-HMAC-CHAP: • Redesign controller authentication to fix security problems • Finish removing Address Mix • Add new Authentication Failed error status. • Use SPSP1 field to identify each message. • Add a lot of error checks. • Merge tables into protocol specification. • Use common hash ids between DH-HMAC-CHAP and SRP in addition to common DH group ids.
1/29/2019	Add names to DH-HMAC-CHAP protocol steps. Add comment on open issue of whether DH-HMAC-CHAP should support controller-only authentication.
2/18/2019	New DH-HMAC-CHAP protocol sequence diagrams. Reset change tracking.
2/19/2019	Remove controller-only authentication support from DH-HMAC-CHAP. Use N-SRP instead. Require N-SRP to use authentication verifier instead of sharing secret. Correct description of Connect Invalid Parameter status. Improve explanation of internal hash function.
2/26/2019	Add Stanford notation for N-SRP and editor's notes on problems and design concerns.
2/27/2019	Add explanation of how order of messages in SRP relates to security of the protocol.
4/23/2019	Add Editor's Note Design sketch for N-SRP redesign, starting on page 24. Two open design issues related to negotiation.

4/25/2019	Results of initial WG discussion added to comments on open design issues. Tentative conclusions are a) to use only one secure hash in N-SRP (remove separate internal-only hash) and b) to restructure negotiation to use <DH group, hash> pairs that correspond to provisioned verifiers to ensure that the results of negotiation correspond to a verifier.
5/9/2019	Initial round of edits, reduce design sketch to reflect initial WG decisions on design issues.
7/7/2020	Initial full rewrite by cds: - Removed N-SRP - Put error handling in authentication messages rather than NVMe-oF response capsules to enable centralized authentication verification - Enabled use of authentication as a way to create pre-shared key material for subsequent secure channel establishment - Made the protocol more robust by introducing a way to relate together the protocol messages - Made future extensibility more modular Conventions: - Text to add in NVMe-oF 1.1 is in blue - Text to delete in NVMe-oF 1.1 is in ~~strikethrough red~~ - New text worth more review is highlighted in magenta
7/21/2020	Incorporated first set of updates: - Replaced DH groups with the TLS 1.3 ones - Added NVMe-oF entity role in the reply computations - Consequentially removed unnecessary anti-replay checks - Updated computations for augmented challenges and TLS PSK Conventions: - Text deleted from the previous version is in ~~strikethrough orange~~ - Text added to the previous version is in green
7/21/2020 concall	Made support for the 3072-bit DH group mandatory
8/18/2020	Incorporated second set of updates: - Defined the ‘key’ and ‘secret’ terminology - Defined a standard ASCII representation of keys/secrets - Improved the strength of the response calculation with HMAC - Allowed reusing the TLS PSK on I/O queue - Improved the payload format - Generalized the TLS_c indication into SC_c Conventions: - Text deleted from the previous version is in ~~strikethrough orange~~ - Text added to the previous version is in green
8/18/2020	- Added support requirements for pair-wise keys - Specified that TLS concatenation should not be used if a TLS PSK was administratively configured - Minor fixes
8/25/2020 concall	- Added support requirements for pair-wise secrets - Minor fixes
8/28/2020	Incorporated comments from Yoni Shternhell Conventions: - Text deleted from the previous version is in ~~strikethrough orange~~ - Text added to the previous version is in green
9/1/2020 concall	- Updated base64 reference - Minor fixes
9/15/2020	Moved to phase 3
10/6/2020	Updated reference to RFC 5996

10/13/2020	- Addressed the 'Two NQNs' issue - Restructured the hash table - Addressed miscellaneous comments from David Black
10/20/2020	- Addressed additional comments
10/27/2020	- Added provision for reauthentication in section 6.2 - Continued to refine section 6.5.7
11/3/2020	- Added provision to section 3.3 to reject a Connect command with HOSTNQN = SUBNQN - Added the NQN of the responding entity to the response computation - Further refined sections 6.5.7 and 6.5.8
11/3/2020 concall	- Concall edits
11/10/2020	- Included both NQNs in the response computation
12/01/2020	- Relaxed secret/key storage requirements in section 6.5.7 - Updated terminology in section 6.5.9
12/03/2020	- Working group call edits
12/15/2020	- Resolved the issue about authentication messages mapping to capsules
12/15/2020 concall	- Editorial fixes
01/28/2021	- Incorporated members' review editorial changes - Incorporated reauthentication specification
03/02/2021	- Incorporated 2nd members' review editorial changes
03/16/2021	- Integrated into the NVMe-oF Specification, Revision 1.1
03/17/2021	- Corrected the reserve bit range in AUTHREQ field.

Incompatible Changes

None.

Description of Specification Changes

Note to the editor: add to the refactored base spec references section the following entry:

INCITS 502-2019, Information technology – SCSI Primary Commands - 5 (SPC-5). Available from http://webstore.ansi.org.

Modify a portion of section 3.3 (Connect Command and Response) as shown below:

3.3 Connect Command and Response

The Connect command is used to create a Submission and Completion Queue pair for an Admin Queue or an I/O Queue. If the Admin Queue is specified, then the Connect command establishes an association between a host and a controller. The fields for the Submission Queue Entry are defined in Figure 19 and the fields for the data portion are defined in Figure 20.

A host that uses a single Host NQN may employ multiple Host Identifiers to designate elements of the host that access an NVM subsystem independently of each other (e.g., physical or logical partitions of the host). Alternatively, a host may employ multiple Host NQN values to cause each element to be treated as a separate host by an NVM subsystem.

If an NVM subsystem supports DH-HMAC-CHAP authentication (refer to section 6), then the Host NQN and the NVM Subsystem NQN parameters in a Connect command are required to be different. If the Host NQN and the NVM Subsystem NQN parameters in a Connect command are identical and the NVM subsystem supports DH-HMAC-CHAP authentication, then a status value of Connect Invalid Host is returned.

…

Figure 22: Connect Response – Dword 0 Value Based on Status Code

<table>
<tr><th>Status Code</th><th>Definition of Dword 0</th></tr>
<tr><td>Successful Completion</td><td>
<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>01:00</td><td>Controller ID (CNTLID): Specifies the controller ID allocated to the host. If a particular controller was specified in the CNTLID field of the Connect command, then this field shall contain the same value.</td></tr>
<tr><td>03:02</td><td>Authentication and Security Requirements (AUTHREQ): Specifies the NVMe in-band authentication and security requirements. The field is bit significant. If ~~no bit is set to '1'~~ all bits are cleared to '0' then no ~~NVMe in-bandauthentication is not required~~ requirements are specified. ~~If a bit is set to '1', then NVMe in-band authentication using the specified protocol is acceptable. If a bit is cleared to '0', then NVMe in-band authentication shall not use the specified protocol.~~
<table>
<tr><th>Bit</th><th>Definition</th></tr>
<tr><td>15:03~~-1~~</td><td>Reserved</td></tr>
<tr><td>02</td><td>If this bit is set to '1', then authentication using NVMe-oF Authentication protocols followed by secure channel establishment is required and bit 01 should be cleared to '0'. If this bit is cleared to '0', then authentication using NVMe-oF Authentication protocols followed by secure channel establishment is not required.</td></tr>
<tr><td>01</td><td>If this bit is set to '1', then authentication using NVMe-oF Authentication protocols is required. If this bit is cleared to '0', then authentication using NVMe-oF Authentication protocols is not required.</td></tr>
<tr><td>00</td><td>~~TCG Security Protocols (refer to TCG Storage Interface Interactions Specification [SIIS])~~ Obsolete.</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

Note to the editor: the new bits should be placed in the field position 01 and 02.

Modify section 6 (Authentication) as shown below:

6 Authentication

NVMe over Fabrics supports both fabric secure channel (that includes authentication) and NVMe in-band authentication. Fabric authentication is part of establishing a fabric secure channel via an NVMe Transport specific protocol that provides authentication, encryption, and integrity checking (e.g., IPsec; see RFC 4301 or TLS; see RFC 8446). NVMe in-band authentication is performed immediately after a Connect command (refer to section 3.3) succeeds using the Authentication Send and Authentication Receive commands (refer to section 3) to tunnel authentication protocol commands between the host and the controller.

Enrollment of the host and controller in an authentication mechanism, including provisioning of authentication credentials to the host and controller, is outside the scope of this specification.

If both fabric secure channel and NVMe in-band authentication are used, the identities for these two instances of authentication may differ for the same NVMe Transport connection. For example, if an iWARP NVMe Transport is used with IPsec as the fabric secure channel technology, the IPsec identities for authentication are associated with the IP network (e.g., DNS host name or IP address), whereas NVMe in-band authentication uses NVMe identities (i.e., Host NQNs). The NVMe Transport binding specification may provide further guidance and requirements on the relationship between these two identities, but determination of which NVMe Transport identities are authorized to be used with which NVMe identities is part of the security policy for the deployed NVM subsystem.

6.1 Fabric Secure Channel

The Transport Requirements field in the Fabrics Discovery Log Page Entry (refer to Figure 38) indicates whether a fabric secure channel shall be used for an NVMe Transport connection to an NVM subsystem. The secure channel mechanism is specific to the type of fabric.

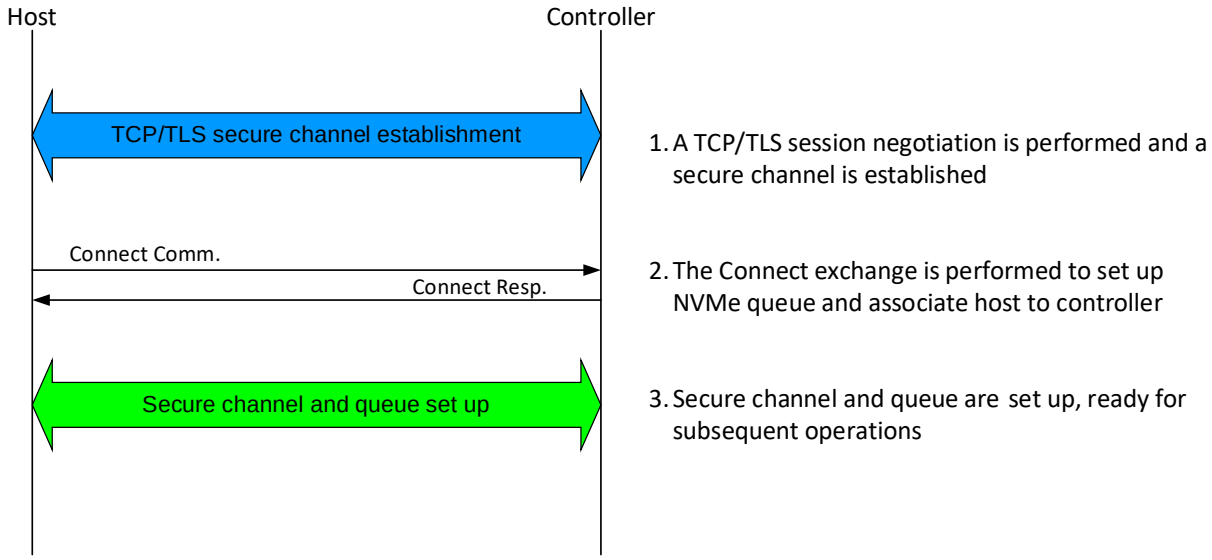
If establishment of a secure channel fails or a secure channel is not established when required by the controller, the resulting errors are fabric-specific and may not be reported to the NVMe layer on the host. Such errors may result in the controller being inaccessible to the host via the NVMe Transport connection on which the authentication failure occurred.

An NVM subsystem that requires use of a fabric secure channel (i.e., as indicated by the TREQ field in the associated Discovery Log entry) shall not allow capsules to be transferred until a secure channel has been established for the NVMe Transport connection.

All Discovery Log Page Entries for an NVM subsystem should report the same value of TREQ to each host. Discovery Log Page Entries for an NVM subsystem may report different values of TREQ to different hosts.

Figure TBD+1 shows an example of secure channel establishment using TLS.

Figure TBD+1 - Example of TLS secure channel establishment



6.2 NVMe In-band Authentication

The Authentication and Security Requirements (AUTHREQ) field in the Connect response capsule (refer to Figure 21) indicates whether NVMe in-band authentication is required.

If one or more of the bits in the AUTHREQ field are set to '1', then the controller requires that the host authenticate on ~~each~~ that queue ~~via one of the indicated security protocols~~ in order to proceed with Fabrics, Admin, and I/O commands. Authentication success is defined by the specific security protocol that is used for authentication. If any command other than Connect, Authentication Send, or Authentication Receive is received prior to authentication success, then the controller shall abort the command with Authentication Required status.

If all bits in the AUTHREQ field are cleared to '0', then the controller does not require the host to authenticate, and the NVM subsystem shall not abort any command with a status value of Authentication Required.

~~A controller shall report the same value of AUTHREQ in the Connect response capsules sent by all of that controller's queues. All controllers in an NVM subsystem should report the same value of AUTHREQ.~~

~~If NVMe in-band authentication succeeds, then:~~

~~1) any supported commands for the associated queue type may be processed; and~~
~~2) if an Authentication Send or an Authentication Receive command is received, then that command shall be aborted with a status value of Command Sequence Error.~~

If NVMe in-band authentication succeeds, then any supported commands for the associated queue type may be processed.

The host may initiate a subsequent authentication transaction at any time for reauthentication purposes. Initiating reauthentication shall not invalidate a prior authentication. If the reauthentication transaction concludes with the controller sending an AUTH_Failure1 message, then the controller shall terminate all commands with a status of Operation Denied and disconnect the NVMe-oF connection. If the reauthentication transaction concludes with the host sending an AUTH_Failure2 message, then the host shall disconnect the NVMe-oF connection.

The state of an in-progress authentication transaction is soft-state. If the subsequent command in an authentication transaction is not received by the controller within a timeout equal to:

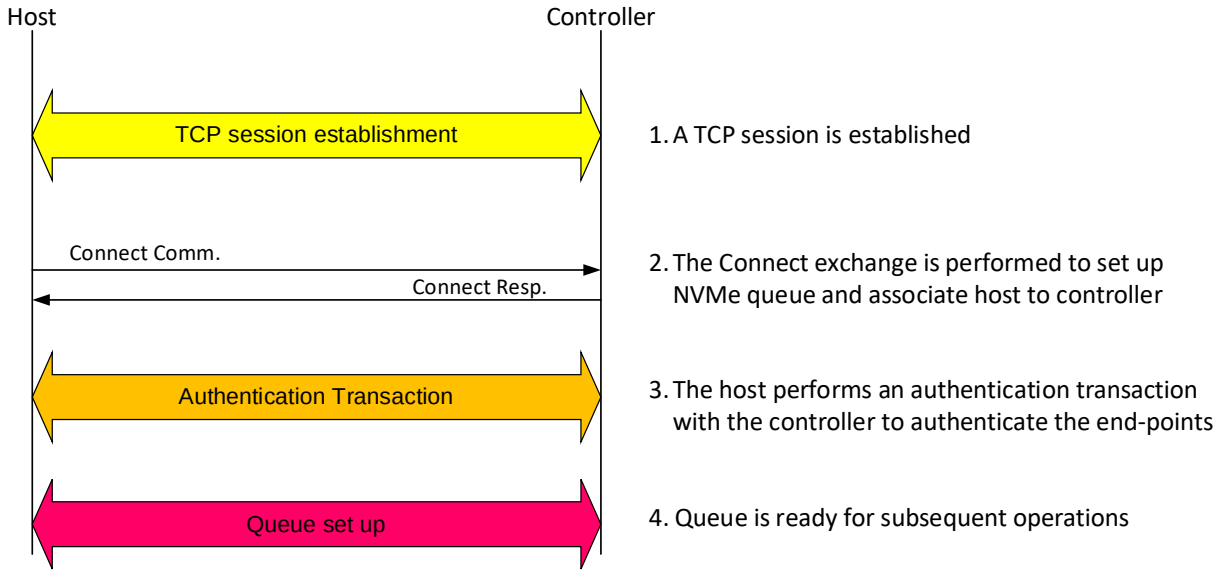
- the Keep Alive Timeout value (refer to Figure 19), if the Keep Alive Timer is enabled; or
- the default Keep Alive Timeout value (i.e., two minutes), if the Keep Alive Timer is disabled;

then the authentication transaction has timed out and the controller should discard the authentication transaction state (including the T_ID value, refer to section 6.4.1).

For an initial authentication, an authentication transaction timeout should be treated as an authentication failure with termination of the transport connection. For reauthentication, an authentication transaction timeout should not be treated as an authentication failure. Authentication commands used to continue that transaction after an authentication transaction timeout should be aborted with a status of Command Sequence Error.

Figure TBD+2 shows an example of authentication transaction for NVMe-oF/TCP.

Figure TBD+2 - Example of authentication transaction for NVMe-oF/TCP



6.2.1 NVMe In-band Authentication Protocol-Specific Requirements

Authentication requirements for security commands are based on the security protocol indicated by the SECP field in the command.

The authentication protocols defined by this specification use the security protocol identifier E9h (assigned to NVMe by SPC-5, a SCSI standard). The messages of the defined authentication protocols are self-identifying, therefore the SPSP0 field and the SPSP1 field of the Authentication Send and Authentication Receive commands shall be set to 01h. Authentication messages are mapped to NVMe-oF command and response pairs. The mapping of authentication messages to the Authentication Send command is shown in Figure TBD+3.

Figure TBD+3 - Mapping of authentication messages to the Authentication Send command

Field[1]	Value
SPSP0	01h
SPSP1	01h
SECP	E9h
TL	Specifies the amount of data to transfer in bytes
NOTES: 1. Refer to section 3.2.	

The mapping of authentication messages to the Authentication Receive command is shown in Figure TBD+4. Security processing requirements associated with the Authentication Receive command (e.g., delays in third-party authentication verification) may result in delays in controller completion of an Authentication Receive command. The host should consider these possible delays associated with the Authentication Receive command.

Figure TBD+4 - Mapping of authentication messages to the Authentication Receive command

Field[1]	Value
SPSP0	01h
SPSP1	01h
SECP	E9h
AL	Specifies the amount of data to transfer in bytes[2]
[1] Refer to section 3.1. [2] The size of the largest authentication message that could be received.	

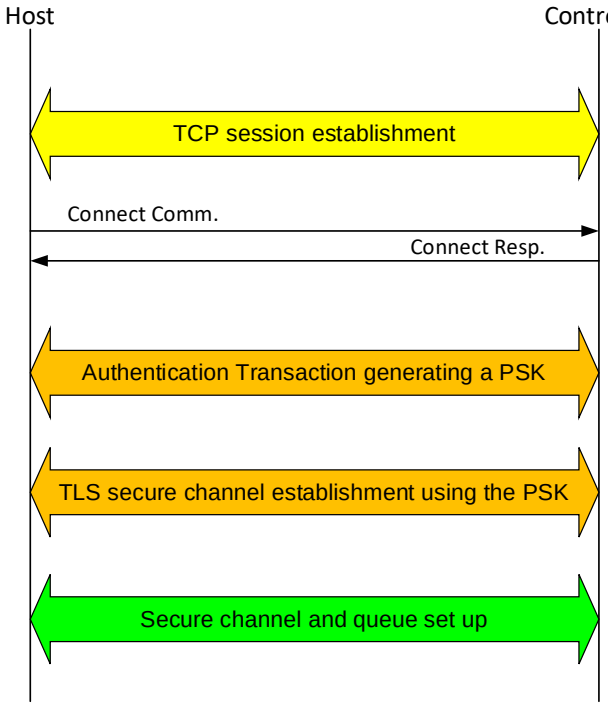
~~6.2.1.1. NVMe In-band Authentication Requirements for the TCG Security Protocols~~

~~For the TCG Security Protocols (i.e., bit 00 is set to '1' in the AUTHREQ field), security commands specifying security protocol values 01h to 06h do not require authentication when used for NVMe in-band authentication. When used for other purposes, in-band authentication of these commands is required. The TCG Storage Interface Interactions Specification (SIIS) and associated specifications specify the subset of the TCG security protocols used for NVMe in-band authentication.~~

6.3 NVMe In-band Authentication Followed by Secure Channel Establishment

It is possible to leverage an authentication transaction to generate shared key material to use as pre-shared key (PSK) to establish a secure channel (e.g., with IPsec or TLS). In this case, the PSK generated to set up a secure channel on the admin queue may be reused to set up additional secure channels on the I/O queues. Figure TBD+5 shows an example of this possibility for TLS.

Figure TBD+5 - Example of TLS secure channel concatenated to an authentication transaction



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

6.4 Common Authentication Messages

6.4.1 AUTH_Negotiate Message

The AUTH_Negotiate message is sent from the host to the controller and is used to indicate the authentication protocols the host is able to use in this authentication transaction and which secure channel protocol, if any, to concatenate to this authentication transaction. The AUTH_Negotiate message format is shown in Figure TBD+6.

Figure TBD+6 - AUTH_Negotiate message format

Bytes	Description
0	**AUTH_TYPE:** 00h (i.e., common messages)
1	**AUTH_ID:** 00h (i.e., AUTH_Negotiate)
3:2	Reserved
5:4	**T_ID:** 16-bit transaction identifier
6	**SC_C:** Secure channel concatenation
7	**NAPD:** Number of authentication protocol descriptors
71:8	Authentication protocol descriptor #1
135:72	Authentication protocol descriptor #2
…	
NAPD*64+7:(NAPD-1)*64+8	Authentication protocol descriptor #NAPD

The SC_C field determines if a secure channel concatenation to the authentication transaction is requested and with which secure channel protocol, as shown in Figure TBD+7.

Figure TBD+7 – Secure channel protocol identifiers

Value	Description	Transport Applicability
00h	No secure channel concatenation	N/A
01h	Secure channel concatenation with TLS (refer to section 6.5.9)	TCP
All others	Reserved	

The AUTH_Negotiate message is structured as a list of 64-byte authentication protocol descriptors to enable extensibility to define additional authentication protocols. Currently only one authentication protocol is defined (i.e., DH-HMAC-CHAP), therefore the AUTH_Negotiate message carries only one authentication protocol descriptor (i.e., NAPD=1). Implementations should support more than one descriptor to enable protocol extensibility. The first byte of an authentication protocol descriptor identifies the specific authentication protocol, as shown in Figure TBD+8.

Figure TBD+8 – Authentication protocol identifiers

Value	Description
01h	DH-HMAC-CHAP (refer to section 6.5)
All others	Reserved

Upon receiving an AUTH_Negotiate message, if the SC_C value indicated by the host does not satisfy the security requirements of the controller (e.g., the host did not request secure channel concatenation, but the controller's security policy requires secure channel concatenation), then the controller shall:

- reply to the AUTH_Negotiate message with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'Secure channel concatenation mismatch'; and
- disconnect the NVMe-oF connection upon transmitting the AUTH_Failure1 message.

Upon receiving an AUTH_Negotiate message, if the protocol descriptors proposed by the host do not satisfy the security requirements of the controller, then the controller shall:

- reply to the AUTH_Negotiate message with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'Authentication protocol not usable'; and
- disconnect the NVMe-oF connection upon transmitting the AUTH_Failure1 message.

6.4.2 AUTH_Failure Messages

The AUTH_Failure1 message is sent from the controller to the host, the AUTH_Failure2 message is sent from the host to the controller. The format of the AUTH_Failure1 message and of the AUTH_Failure2 message is shown in Figure TBD+9.

Figure TBD+9 – AUTH_Failure1 and AUTH_Failure2 message format

Bytes	Description
0	**AUTH_TYPE:** 00h (i.e., common messages)
1	**AUTH_ID:** • F0h (i.e., AUTH_Failure2) • F1h (i.e., AUTH_Failure1)
3:2	Reserved
5:4	**T_ID:** 16-bit transaction identifier
6	Reason code
7	Reason code explanation

The AUTH_Failure reason codes are listed in Figure TBD+10.

Figure TBD+10 – AUTH_Failure reason codes

Value	Description
01h	**Authentication failure:** The authentication transaction failed
All others	Reserved

The AUTH_Failure reason code explanations are listed in Figure TBD+11.

Figure TBD+11 – AUTH_Failure reason code explanations

Value	Description
01h	**Authentication failed:** Authentication of the involved host or NVM subsystem failed
02h	**Authentication protocol not usable:** The protocol descriptors proposed by the host do not satisfy the security requirements of the controller (refer to section 6.4.1)
03h	**Secure channel concatenation mismatch:** The SC_C value indicated by the host does not satisfy the security requirements of the controller (refer to section 6.4.1)
04h	**Hash function not usable:** The HashIDList proposed by the host does not satisfy the security requirements of the controller (refer to section 6.5.2)
05h	**DH group not usable:** The DHgIDList proposed by the host does not satisfy the security requirements of the controller (refer to section 6.5.2)
06h	**Incorrect payload:** The payload of the received message is not correct
07h	**Incorrect protocol message:** The received message is not the expected next message in the authentication protocol sequence
All others	Reserved

6.4.3 Mapping of Common Authentication Messages to Authentication Commands

The AUTH_Negotiate message and the AUTH_Failure2 message are sent from the host to the controller, therefore they are mapped to the Authentication Send command. The AUTH_Failure1 message is sent from the controller to the host, therefore it is mapped to the Authentication Receive command.

6.5 DH-HMAC-CHAP Protocol

6.5.1 Protocol Operations

DH-HMAC-CHAP is a key based Authentication and key management protocol that uses the Challenge Handshake Authentication Protocol (CHAP, see RFC 1994) enhanced to use the Hashed Message Authentication Code (HMAC) mechanism (see RFC 2104) with stronger hash functions and augmented with an optional Diffie-Hellman (DH) exchange (see RFC 2631, clause 2.2.1). DH-HMAC-CHAP provides bidirectional or unidirectional Authentication between a host and a controller.

The Diffie-Hellman part of the protocol is optional. When the Diffie-Hellman part of the protocol is not used, DH-HMAC-CHAP is referred to as HMAC-CHAP. If insufficiently random keys are used (refer to section 6.5.7), HMAC-CHAP potentially allows a passive eavesdropper to discover the key through an off-line dictionary attack, so its usage should be minimized. DH-HMAC-CHAP provides strong protection from passive eavesdroppers. However, an active attacker could reduce the operation of this protocol so that only HMAC-CHAP is used, and as a result gain sufficient information to mount an off-line dictionary attack on the HMAC-CHAP key.

An implementation that supports DH-HMAC-CHAP authentication shall support DH-HMAC-CHAP with a NULL DH exchange. All implementations of DH-HMAC-CHAP shall be configurable to require a DH exchange (i.e., to not use HMAC-CHAP).

In order to authenticate with the DH-HMAC-CHAP protocol, each host and NVM subsystem shall be provided with a DH-HMAC-CHAP key that is associated with the entity's NQN. Two entities may impersonate one another if they have the same key, therefore when the assigned keys are not different for each entity there is a security vulnerability (refer to section 6.5.7).

To authenticate another entity, an entity is required to either:

a) know the key associated with the entity to be authenticated; or
b) rely on a third party that knows the key to verify the authentication.

An example of a DH-HMAC-CHAP authentication transaction is shown in Figure TBD+12, with the notation shown in Figure TBD+13. The DH-HMAC-CHAP_Success2 message that is shown as a dashed line is used only for bidirectional authentication.

Figure TBD+12 - Example of DH-HMAC-CHAP authentication transaction

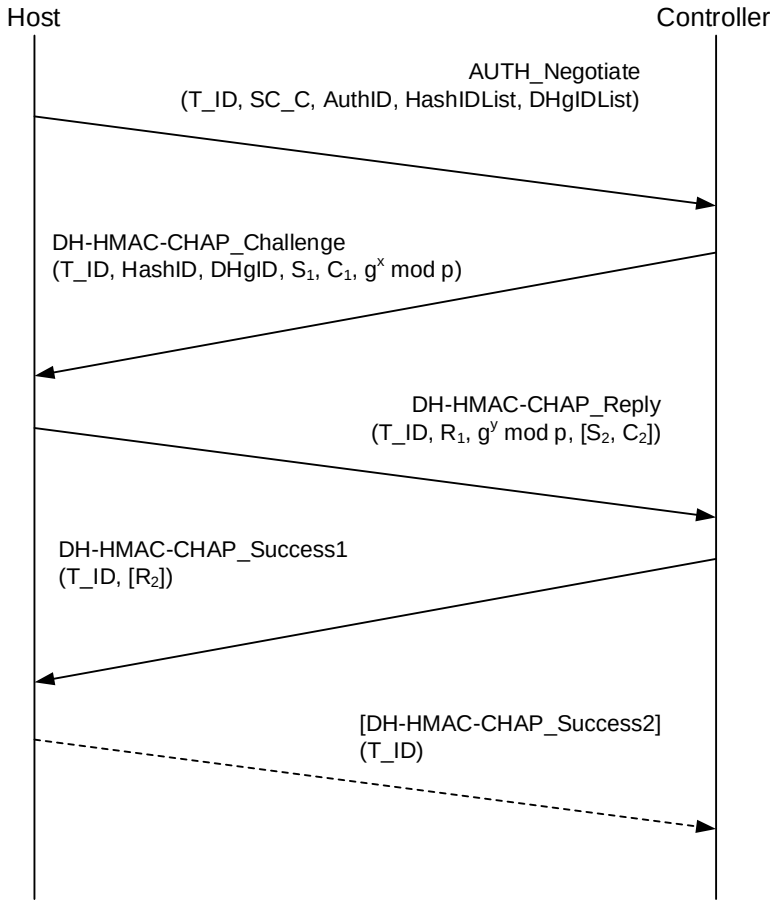


Figure TBD+13 - Mathematical notations for DH-HMAC-CHAP

Symbols	Description
NQN_c, NQN_h	NQN of the NVM subsystem that contains the controller and NQN of the host
K_c, K_h	DH-HMAC-CHAP key of the NVM subsystem that contains the controller and DH-HMAC-CHAP key of the host
p, g	Modulus (p) and generator (g) of the chosen DH group (refer to Figure TBD+16)
x, y	Random numbers used as exponents in a DH exchange
C_1, C_2	Random challenge values
C_{a1}, C_{a2}	Augmented challenge values
S_1, S_2	32-bit sequence numbers
R_1, R_2	Reply values
T_ID	Authentication transaction identifier
SC_C	Secure channel concatenation indication
H( )	One-way hash function (refer to Figure TBD+15)
HMAC(K, Str)	HMAC function (see RFC 2104) with key K on string Str using hash function H( )
||	Concatenation operation
K_S	Session key

When used with a non-NULL DH exchange, the DH-HMAC-CHAP protocol is able to generate a session key K_S to be used to establish a TLS session between host and controller (refer to section 6.5.9).

For an NVM subsystem, the controller is the entity running the protocol, using the identity and credentials of the NVM subsystem. The DH-HMAC-CHAP protocol proceeds in the following order:

1) The authentication transaction shall begin with the host sending the common AUTH_Negotiate message to negotiate the authentication protocol to use and its associated parameters (refer to section 6.4.1). The AUTH_Negotiate message carries the transaction identifier (T_ID) for the entire authentication transaction and the list of authentication protocol descriptors for the authentication protocols that may be used in this authentication transaction. For DH-HMAC-CHAP, the authentication protocol descriptor includes the list of hash functions (HashIDList) and Diffie-Hellman group identifiers (DHgIDList) that may be used in this authentication protocol transaction.

2) If the parameters of the received DH-HMAC-CHAP protocol descriptor are compatible with the controller's policies, then the controller shall reply with a DH-HMAC-CHAP_Challenge message (refer to section 6.5.3) carrying the same transaction identifier value (T_ID) received in the AUTH_Negotiate message, the identifiers of the hash function (HashID) and the DH group (DHgID) selected for use among the ones proposed by the host in the AUTH_Negotiate message, a sequence number (S_1), a random challenge value (C_1), and the DH exponential (g^x mod p). If the controller selects a NULL DH group identifier, then the DH portion of the DH-HMAC-CHAP protocol shall not be used, and the protocol reduces to a HMAC-CHAP transaction.

3) If the received DH-HMAC-CHAP_Challenge message is valid, then the host shall send a DH-HMAC-CHAP_Reply message (refer to section 6.5.4) carrying the same transaction identifier value (T_ID), the response R_1 to the challenge value C_1, and its own DH exponential (g^y mod p). The DH Value Length shall be cleared to 0h if the controller has sent a NULL DH group identifier in the DH-HMAC-CHAP_Challenge message. If bidirectional authentication is requested, then the DH-HMAC-CHAP_Reply message shall carry also a sequence number S_2 and a random challenge value C_2 that differs from the challenge value C_1 received in the DH-HMAC-CHAP_Challenge message.

4) If the authentication verification by the controller succeeds, then the controller shall reply with a DH-HMAC-CHAP_Success1 message (refer to section 6.5.5) carrying the same transaction identifier value (T_ID). If bidirectional authentication was requested, then the DH-HMAC-CHAP_Success1 message shall also carry the response R_2 to the challenge value C_2. If the authentication verification fails, then the controller shall send an AUTH_Failure1 message and disconnect the NVMe-oF connection upon transmitting it.

5) The authentication transaction ends here, unless bidirectional authentication has been requested. In this case, as shown by the dashed arrow in Figure TBD+12, if the authentication verification by the host succeeds, then the host shall send a DH-HMAC-CHAP_Success2 message (refer to section 6.5.6) carrying the same transaction identifier value (T_ID). If the authentication verification fails, then the host shall send an AUTH_Failure2 message and disconnect the NVMe-oF connection upon transmitting it.

If the controller receives a message that is not the expected next message in the DH-HMAC-CHAP protocol sequence, then the controller shall:

- reply with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'Incorrect protocol message'; and
- disconnect the NVMe-oF connection upon transmitting the AUTH_Failure1 message.

If the host receives a message that is not the expected next message in the DH-HMAC-CHAP protocol sequence, then the host shall:

- reply with an AUTH_Failure2 message having reason code 'Authentication failure' and reason code explanation 'Incorrect protocol message'; and
- disconnect the NVMe-oF connection upon transmitting the AUTH_Failure2 message.

The payload format of a message shall be validated before performing any other security computation.

6.5.2 DH-HMAC-CHAP Authentication Protocol Descriptor

The authentication protocol descriptor for DH-HMAC-CHAP (refer to section 6.4.1) is shown in Figure TBD+14.

Figure TBD+14 - Authentication protocol descriptor for DH-HMAC-CHAP

Bytes	Description
0	**AuthID:** Authentication protocol identifier (01h for DH-HMAC-CHAP)
1	Reserved
2	**HashIDList Length (HALEN):** Number of hash function identifiers (1 to 30)
3	**DHgIDList Length (DHLEN):** Number of Diffie-Hellman group identifiers (1 to 30)
3+HALEN:4	**HashIDList:** Array of hash function identifiers (one byte per identifier)
33:4+HALEN	Padding bytes cleared to 0h, if present
33+DHLEN:34	**DHgIDList:** Array of Diffie-Hellman Group identifiers (one byte per identifier)
63:34+DHLEN	Padding bytes cleared to 0h, if present

The one-way hash functions used by DH-HMAC-CHAP are shown in Figure TBD+15.

Figure TBD+15 - DH-HMAC-CHAP hash function identifiers

Identifier	Hash Function	Hash Length (bytes)	Hash Block Size[1] (bytes)	Reference
00h	Reserved			
01h	SHA-256	32	64	RFC 6234
02h	SHA-384	48	128	RFC 6234
03h	SHA-512	64	128	RFC 6234
04h-DFh	Reserved			
E0h-FEh	Vendor specific			
FFh	Reserved			
NOTES: 1. The hash block size is used by the HMAC calculation				

The SHA-256 hash function shall be supported.

Upon receiving an AUTH_Negotiate message, if the HashIDList proposed by the host does not satisfy the security requirements of the controller (e.g., the host proposed SHA-256, but the controller's security policy requires a SHA-384 hash), then the controller shall:

- reply to the AUTH_Negotiate message with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'Hash function not usable'; and
- disconnect the NVMe-oF connection upon transmitting the AUTH_Failure1 message.

The Diffie-Hellman (DH) groups used by DH-HMAC-CHAP are shown in Figure TBD+16.

Figure TBD+16 – DH-HMAC-CHAP Diffie-Hellman group identifiers

Identifier	DH group size	Generator (*g*)	Modulus (*p*) and Reference
00h	NULL	N/A	N/A
01h	2048-bit	2	see RFC 7919
02h	3072-bit	2	see RFC 7919
03h	4096-bit	2	see RFC 7919
04h	6144-bit	2	see RFC 7919
05h	8192-bit	2	see RFC 7919
06h-DFh	Reserved		
E0h-FEh	Vendor specific		
FFh	Reserved		

The 00h identifier indicates that no Diffie-Hellman exchange is performed, which reduces the DH-HMAC-CHAP protocol to the HMAC-CHAP protocol.

The 2048-bit DH group and the 3072-bit DH group shall be supported. A mechanism shall be provided to disable (i.e., prohibit) use of the 2048-bit DH group.

Upon receiving an AUTH_Negotiate message, if the DHgIDList proposed by the host does not satisfy the security requirements of the controller (e.g., the host proposed only the NULL DH group, but the controller's security policy requires a DH group whose size is 3072-bit or larger), then the controller shall:

- reply to the AUTH_Negotiate message with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'DH group not usable'; and

- disconnect the NVMe-oF connection upon transmitting the AUTH_Failure1 message.

6.5.3 DH-HMAC-CHAP_Challenge Message

The DH-HMAC-CHAP_Challenge message is sent for the controller to the host. The format of the DH-HMAC-CHAP_Challenge message is shown in Figure TBD+17.

Figure TBD+17 - DH-HMAC-CHAP_Challenge message format

Bytes	Description
0	**AUTH_TYPE:** 01h (i.e., DH-HMAC-CHAP)
1	**AUTH_ID:** 01h (i.e., DH-HMAC-CHAP_Challenge)
3:2	Reserved
5:4	**T_ID:** 16-bit transaction identifier
6	**Hash Length (HL):** Length in bytes of the selected hash function
7	Reserved
8	**HashID:** Identifier of selected hash function
9	**DHgID:** Identifier of selected Diffie-Hellman group
11:10	**DH Value Length (DHVLEN):** Length in bytes of DH value. If no DH value is included in the message, then this field is cleared to 0h
15:12	**Sequence Number (SEQNUM):** Sequence number S_1
15+HL:16	**Challenge Value (CVAL):** Challenge C_1
15+HL+DHVLEN:16+HL	**DH Value (DHV):** DH exponential g^x mod p. This field is not present (i.e., the CVAL field is the last field in the message) if DHVLEN is cleared to 0h

Hash Length (HL): Shall be set to the length in bytes of the selected hash function, as specified in Figure TBD+15.

HashID: Shall be set to the hash function identifier (refer to Figure TBD+15) selected for this authentication transaction among those proposed in the DH-HMAC-CHAP protocol descriptor in the AUTH_Negotiate message. The controller shall select a hash function in accord with its applicable policy.

DHgID: Shall be set to the DH group identifier (refer to Figure TBD+16) selected for this authentication transaction among those proposed in the DH-HMAC-CHAP protocol descriptor in the AUTH_Negotiate message. The controller shall select a DH group identifier in accord with its applicable policy. If this field is cleared to 0h, the DH portion of the DH-HMAC-CHAP protocol shall not be performed in this authentication transaction.

DH Value Length (DHVLEN): Diffie-Hellman exponential length. This length shall be a multiple of 4. If the DH group identifier is cleared to 0h (i.e., NULL DH exchange), this field shall be cleared to 0h. Otherwise, it shall be set to the length in bytes of the DH Value.

Sequence Number (SEQNUM): 32-bit sequence number S_1. A random non-zero value shall be used as the initial value. The sequence number is incremented modulo 2^{32} after each use, except that the value 0h is skipped (i.e., incrementing the value FFFF FFFFh results in the value 0000 0001h).

Challenge Value (CVAL): Shall be set to a random challenge value C_1 (refer to section 6.5.7). Each challenge value should be unique and unpredictable, since repetition of a challenge value in conjunction with the same key may reveal information about the key or the correct response to this challenge. The algorithm for generating the challenge value is outside the scope of this specification. Randomness of the challenge value is crucial to the security of the protocol (refer to section 6.5.7). The CVAL length is the same as the length of the selected hash function (i.e., HL).

DH Value (DHV): Diffie-Hellman exponential. If the DH Value Length is cleared to 0h, this field is not present. The DH value shall be set to the value of g^x mod p, where x is a random number selected by the controller that shall be at least 256 bits long (refer to section 6.5.7) and p and g shall have the values indicated in Figure TBD+16, based on the selected DH group identifier.

Upon receiving a DH-HMAC-CHAP_Challenge message, if:

- the Hash Length (HL) does not match the value specified in Figure TBD+15 for the selected hash function;
- the Sequence Number (SEQNUM) is cleared to 0h;
- DHgID is non-zero and the DH Value Length (DHVLEN) is cleared to 0h; or
- DHgID is non-zero and the DH Value (DHV) is 0, 1, or p-1;

then the host shall:

- reply with an AUTH_Failure2 having reason code 'Authentication failure' and reason code explanation 'Incorrect payload'; and
- disconnect the NVMe-oF connection.

6.5.4 DH-HMAC-CHAP_Reply Message

The DH-HMAC-CHAP_Reply message is sent from the host to the controller. The host may request authentication of the controller to enable bidirectional authentication, by including a DH-HMAC-CHAP challenge value C_2 in this message. The challenge value C_2 shall be different from the challenge value C_1 received in the DH-HMAC-CHAP_Challenge message.

The format of the DH-HMAC-CHAP_Reply message is shown in Figure TBD+18.

Figure TBD+18 - DH-HMAC-CHAP_Reply message format

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>0</td><td>AUTH_TYPE: 01h (i.e., DH-HMAC-CHAP)</td></tr>
<tr><td>1</td><td>AUTH_ID: 02h (i.e., DH-HMAC-CHAP_Reply)</td></tr>
<tr><td>3:2</td><td>Reserved</td></tr>
<tr><td>5:4</td><td>T_ID: 16-bit transaction identifier</td></tr>
<tr><td>6</td><td>Hash Length (HL): Length in bytes of the selected hash function</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>8</td><td>Challenge Valid (CVALID):
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00h</td><td>The Challenge Value is not valid</td></tr>
<tr><td>01h</td><td>The Challenge Value is valid</td></tr>
<tr><td>All others</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>9</td><td>Reserved</td></tr>
<tr><td>11:10</td><td>DH Value Length (DHVLEN): Length in bytes of DH value. If no DH value is included in the message, then this field is cleared to 0h</td></tr>
<tr><td>15:12</td><td>Sequence Number (SEQNUM): Sequence number S_2</td></tr>
<tr><td>15+HL:16</td><td>Response Value (RVAL): Response R_1</td></tr>
<tr><td>15+2*HL:16+HL</td><td>Challenge Value (CVAL): Challenge C_2, if valid (i.e., if the CVALID field is set to 01h), cleared to 0h otherwise</td></tr>
<tr><td>15+2*HL+DHVLEN:16+2*HL</td><td>DH Value (DHV): DH exponential g^y mod p. This field is not present (i.e., the CVAL field is the last field in the message) if DHVLEN is cleared to 0h</td></tr>
</table>

Hash Length (HL): Shall be set to the length in bytes of the selected hash function, as specified in Figure TBD+15.

Challenge Valid: If the host does not require bidirectional authentication or no establishment of a secure channel after unidirectional authentication is sought (refer to section 6.5.8), this field shall be cleared to 0h. Otherwise, this field shall be set to 01h.

DH Value Length (DHVLEN): Diffie-Hellman exponential length. This length shall be a multiple of 4. If the DH group identifier is cleared to 0h (i.e., NULL DH exchange), this field shall be cleared to 0h. Otherwise, it shall be set to the length in bytes of the DH Value.

Sequence Number (SEQNUM): 32-bit sequence number S_2. A random non-zero value shall be used as the initial value. The sequence number is incremented modulo 2^{32} after each use, except that the value 0h is skipped (i.e., incrementing the value FFFF FFFFh results in the value 0000 0001h). The value 0h is used to indicate that bidirectional authentication is not performed, but a challenge value C_2 is carried in order to generate a pre-shared key (PSK) for subsequent establishment of a secure channel (refer to section 6.5.8).

Response Value (RVAL): DH-HMAC-CHAP response value R_1. The value of R_1 is computed using the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message, and the augmented challenge C_{a1}. If the NULL DH group has been selected, the augmented challenge C_{a1} is equal to the challenge C_1 received from the controller (i.e., $C_{a1} = C_1$). If a non-NULL DH group has been selected, the augmented challenge is computed applying the HMAC function using the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message with the hash of the ephemeral DH key resulting from the combination of the random value y selected by the host with the DH exponential (i.e., g^x mod p) received from the controller as HMAC key (see RFC 2104) to the challenge C_1 (i.e., $C_{a1} = HMAC(H((g^x \text{ mod } p)^y \text{ mod } p), C_1) = HMAC(H(g^{xy} \text{ mod } p), C_1)$). The value of R_1 shall be computed applying the HMAC function using the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message with key K_h as HMAC key to the concatenation of the augmented challenge C_{a1}, the sequence number S_1, the transaction identifier T_ID, the secure channel concatenation indication SC_C sent in the AUTH_Negotiate message, the eight ASCII characters ”HostHost” to indicate the host is computing the reply, the host NQN not including the null terminator, a 00h byte, and the NVM subsystem NQN not including the null terminator (i.e., $R_1 = HMAC(K_h, C_{a1} \| S_1 \| T\_ID \| SC\_C \| \text{”HostHost”} \| NQN_h \| 00h \| NQN_c)$). Using C language notation:

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

Challenge Value (CVAL): Shall be set to a random challenge value C_2 (refer to section 6.5.7). Each challenge value should be unique and unpredictable, since repetition of a challenge value in conjunction with the same key may reveal information about the key or the correct response to this challenge. The algorithm for generating the challenge value is outside the scope of this specification. Randomness of the challenge value is crucial to the security of the protocol (refer to section 6.5.7). The CVAL length is the same as the length of the selected hash function (i.e., HL).

DH Value (DHV): Diffie-Hellman exponential. If the DH Value Length is cleared to 0h, this field is not present. The DH Value shall be set to the value of g^y mod p, where y is a random number selected by the host that shall be at least 256 bits long (refer to section 6.5.7) and p and g shall have the values indicated in Figure TBD+16, based on the selected DH group identifier.

Upon receiving a DH-HMAC-CHAP_Reply message, if:

- the Hash Length (HL) does not match the value specified in Figure TBD+15 for the selected hash function;
- DHgID is non-zero and the DH Value Length (DHVLEN) is cleared to 0h; or
- DHgID is non-zero and the DH Value (DHV) is 0, 1, or p-1;

then the controller shall:

- reply with an AUTH_Failure1 message having reason code ‘Authentication failure’ and reason code explanation ‘Incorrect payload’; and
- disconnect the NVMe-oF connection.

In addition, the controller shall:

- check the challenge value C_2, if the Challenge Valid field is set to 01h, to verify it is different from the challenge value C_1 the controller previously sent. If C_2 is equal to C_1, the controller shall:
  - reply with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'Authentication failed'; and
  - disconnect the NVMe-oF connection; and
- verify the response value R_1 using the negotiated hash function. If verification of the response value R_1 does not succeed, the controller shall:
  - reply with an AUTH_Failure1 message having reason code 'Authentication failure' and reason code explanation 'Authentication failed'; and
  - disconnect the NVMe-oF connection.

If verification of the response value R_1 succeeds, the host has been authenticated and the controller shall continue with a DH-HMAC-CHAP_Success1 message.

6.5.5 DH-HMAC-CHAP_Success1 Message

The DH-HMAC-CHAP_Success1 message is sent from the controller to the host and indicates that the controller has successfully authenticated the host. The format of the DH-HMAC-CHAP_Success1 message is shown in Figure TBD+19.

Figure TBD+19 - DH-HMAC-CHAP_Success1 message format

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>0</td><td>AUTH_TYPE: 01h (i.e., DH-HMAC-CHAP)</td></tr>
<tr><td>1</td><td>AUTH_ID: 03h (i.e., DH-HMAC-CHAP_Success1)</td></tr>
<tr><td>3:2</td><td>Reserved</td></tr>
<tr><td>5:4</td><td>T_ID: 16-bit transaction identifier</td></tr>
<tr><td>6</td><td>Hash Length (HL): Length in bytes of the selected hash function</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>8</td><td>Response Valid (RVALID):
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00h</td><td>The Response Value is not valid</td></tr>
<tr><td>01h</td><td>The Response Value is valid</td></tr>
<tr><td>All others</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>15:9</td><td>Reserved</td></tr>
<tr><td>15+HL:16</td><td>Response Value (RVAL): Response R_2, if valid (i.e., if the RVALID field is set to 01h), cleared to zero otherwise</td></tr>
</table>

Hash Length (HL): Shall be set to the length in bytes of the selected hash function, as specified in Figure TBD+15.

Response Valid: If the host did not request authentication of the controller (i.e., bidirectional authentication) this field shall be cleared to 0h to indicate that no response is conveyed (i.e., the Response Value field is not valid). If the host did request authentication of the controller, this field shall be set to 01h.

Response Value (RVAL): DH-HMAC-CHAP response value R_2. The value of R_2 is computed using the hash function H( ) selected by the HashID parameter of the DH-HMAC-CHAP_Challenge message, and the augmented challenge C_{a2}. If the NULL DH group has been selected, the augmented challenge C_{a2} is equal to the challenge C_2 received from the host (i.e., $C_{a2} = C_2$). If a non-NULL DH group has been selected, the augmented challenge is computed applying the HMAC function using the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message with the hash of the ephemeral DH key resulting from the combination of the random value x selected by the controller with the DH exponential (i.e., g^y mod p) received from the host as HMAC key (see RFC 2104) to the challenge C_2 (i.e., C_{a2} = HMAC(H((g^y mod p)^x mod p), C_2) = HMAC(H(g^{xy} mod p)), C_2). The value of R_2 shall be computed applying

the HMAC function using the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message with key K_c as HMAC key to the concatenation of the augmented challenge C_{a2}, the sequence number S_2, the transaction identifier T_ID, the secure channel concatenation indication SC_C received in the AUTH_Negotiate message, the ten ASCII characters ”Controller” to indicate the controller is computing the reply, the NVM subsystem NQN not including the null terminator, a 00h byte, and the host NQN not including the null terminator (i.e., R_2 = HMAC(K_c, C_{a2} || S_2 || T_ID || SC_C || ”Controller” || NQN_c || 00h || NQN_h)). Using C language notation:

C_{a2} = (DHgID == 00h) ? C_2 : HMAC(H((g^y mod p)^x mod p)), C_2)
R_2 = HMAC(K_c, C_{a2} || S_2 || T_ID || SC_C || ”Controller” || NQN_c || 00h || NQN_h)

Upon receiving a DH-HMAC-CHAP_Success1 message:

- if the Hash Length (HL) does not match the value specified in Figure TBD+15 for the selected hash function, the host shall:
  - reply with an AUTH_Failure2 message having reason code ‘Authentication failure’ and reason code explanation ‘Incorrect payload’; and
  - disconnect the NVMe-oF connection; and
- if the Response Valid field is set to 01h, the host shall verify the response value R_2 using the negotiated hash function and DH group. If verification of the response value R_2 does not succeed, the host shall:
  - reply with an AUTH_Failure2 message having reason code ‘Authentication failure’ and reason code explanation ‘Authentication failed’; and
  - disconnect the NVMe-oF connection.

If verification of the response value R_2 succeeds, the controller has been authenticated and the host shall continue with a DH-HMAC-CHAP_Success2 message.

6.5.6 DH-HMAC-CHAP_Success2 Message

The DH-HMAC-CHAP_Success2 message is sent from the host to the controller and indicates that the host has successfully authenticated the controller. The format of the DH-HMAC-CHAP_Success2 message is shown in Figure TBD+20.

Figure TBD+20 - DH-HMAC-CHAP_Success2 message format

Bytes	Description
0	**AUTH_TYPE:** 01h (i.e., DH-HMAC-CHAP)
1	**AUTH_ID:** 04h (i.e., DH-HMAC-CHAP_Success2)
3:2	Reserved
5:4	**T_ID:** 16-bit transaction identifier
15:6	Reserved

6.5.7 DH-HMAC-CHAP Security Requirements

In order to authenticate with the DH-HMAC-CHAP protocol, each host or controller uses a DH-HMAC-CHAP key that is associated with the entity’s NQN. A DH-HMAC-CHAP key is unidirectional (i.e., used only for one direction of an authentication transaction). A DH-HMAC-CHAP key should not be associated with more than one NQN as this opens security vulnerabilities. All DH-HMAC-CHAP implementations should check for use of the same key with more than one NQN and should generate an administrative warning if this situation occurs (e.g., as a result of configuring a DH-HMAC-CHAP key to verify authentication of another entity).

The DH-HMAC-CHAP key is derived from an administratively configured secret (refer to section 6.5.8). Each host and NVM subsystem shall support:

- transforming the provided secret into a key applying the HMAC function using the hash function specified in the secret representation (refer to section 6.5.8) with the secret as HMAC key to the

concatenation of its own NQN not including the null terminator and the seventeen ASCII characters “NVMe-over-Fabrics” (i.e., key = HMAC(secret, NQN || ”NVMe-over-Fabrics”)). This transformation ensures the resulting key is uniquely associated with the entity identified by the NQN; and
- using the provided secret as a key. This is intended for use with key management solutions able to ensure that key is uniquely associated with the entity identified by the NQN.

NVM subsystems should support the ability to use a different NVM subsystem key with each host. Hosts should support the ability to use a different host key with each NVM subsystem. NVM subsystems should support the ability to use a different NVM subsystem secret with each host. Hosts should support the ability to use a different host secret with each NVM subsystem.

If an implementation of NVMe over Fabrics is capable of functioning as both a host and an NVM subsystem, then that implementation shall use either:

- one NQN for the host functionality and a different NQN for the NVM subsystem functionality; or
- one NQN for both host functionality and NVM subsystem functionality.

DH-HMAC-CHAP implementations may reuse a DH exponential (e.g., g^x mod p or g^y mod p). The primary risk in allowing reuse of a DH exponential is replay of a prior authentication sequence based on the attacker reusing the other exponential. For DH-HMAC-CHAP, replay is prevented with extremely high probability by the requirement that all challenges be randomly generated. See section 2.12 of RFC 7296 for guidance on DH exponential reuse.

The security of the DH-HMAC-CHAP protocol requires secrets, challenges, and DH exponents (i.e., x and y) to be generated from actual randomness. For a discussion of randomness and sources of randomness, refer to RFC 4086.

Implementations shall use a cryptographic random number generator that should be seeded with at least 256 bits of entropy to generate random numbers for this protocol. The secret provisioning mechanism for each host and controller is outside of scope of this specification. For instance, secrets could be provisioned via an encrypted HTTPS-based connection.

6.5.8 Secret Representation

In order to facilitate provisioning, management, and interchange (e.g., copy & paste in an administrative configuration tool) of secrets, all NVMe-oF entities shall support the following ASCII representation of secrets:

```
DHHC-1:xx:<Base64 encoded string>:
```

Where:

- ”DHHC-1” indicates this is a version 1 representation of a secret for the DH-HMAC-CHAP protocol;
- ‘:’ is used both as a separator and a terminator;
- xx indicates the hash function to be used to transform the secret in key (refer to section 6.5.7), encoded as the ASCII representation of the hexadecimal value specified in Figure TBD+15 (e.g., the two ASCII characters “01” indicate SHA-256). The two ASCII characters “00” indicate no transform (i.e., use the secret as a key); and
- The Base64 (see RFC 4648) string encodes the secret (32, 48, or 64 bytes binary) followed by the CRC-32 (see RFC 1952) of the secret (4 bytes binary).

As an example, the 32-byte secret:
`89AEB31A 874EAF84 841B4673 6B0DFDF2 BA58D30A A2A545A3 E235A352 1E07594Ch`
is represented as: “`DHHC-1:00:ia6zGodOr4SEG0Zzaw398rpY0wqipUWj4jWjUh4HWUz6aQ2n:`”
when intended to be used as a key without transform.

When provided with a secret in this format, NVMe-oF entities shall verify the validity of the provided secret by computing the CRC-32 value of the secret and checking the computed value with the provided value. If they do not match, then the secret shall not be used.

6.5.9 Generated PSK for TLS

When used with a non-NULL DH exchange, the DH-HMAC-CHAP protocol is able to generate a session key K_S used to generate a pre-shared key (PSK) to establish a secure channel session with the TLS protocol between host and controller. A TLS session is concatenated to an authentication transaction when the SC_C indication is set to 01h in the AUTH_Negotiate message. A TLS session should not be concatenated to an authentication transaction if the involved host and controller are administratively configured with a PSK for use with each other. In this case, host and controller should just establish a TLS session based on the configured PSK.

The session key K_S shall be computed from the ephemeral DH key (i.e., g^{xy} mod p) generated during the DH-HMAC-CHAP transaction by applying the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message (i.e., $K_S = H(g^{xy} \text{ mod } p)$). The size of the session key K_S is determined by the selected hash function, as shown in Figure TBD+15. Specifically:

- The host computes K_S as the hash of the ephemeral DH key resulting from the combination of the random value y selected by the host with the DH exponential (i.e., g^x mod p) received from the controller (i.e., $K_S = H((g^x \text{ mod } p)^y \text{ mod } p) = H(g^{xy} \text{ mod } p)$).
- The controller computes K_S as the hash of the ephemeral DH key resulting from the combination of the random value x selected by the controller with the DH exponential (i.e., g^y mod p) received from the host (i.e., $K_S = H((g^y \text{ mod } p)^x \text{ mod } p) = H(g^{xy} \text{ mod } p)$).

The generated PSK for TLS shall be computed applying the HMAC function using the hash function H( ) selected by the HashID parameter in the DH-HMAC-CHAP_Challenge message with the session key K_S as key to the concatenation of the two challenges C_1 and C_2 (i.e., generated PSK = HMAC(K_S, C_1 || C_2)). The generated PSK used to set up a TLS secure channel on the admin queue may be reused to set up additional TLS secure channels on the I/O queues (refer to section 7.4.9.4). The lifetime of this generated PSK should be no more than ten minutes; this requires authentication for I/O queues created after this time.

Note to the editor: 7.4.9.4 is a section of TP 8011.

The host may request secure channel concatenation with the TLS protocol by setting the SC_C indication in the AUTH_Negotiate message to 01h while performing only unidirectional authentication. In this case the host shall still send a challenge value C_2 to the controller and clear the sequence number S_2 to 0h to indicate that controller authentication is not requested.

6.5.10 Mapping of DH-HMAC-CHAP Messages to Authentication Commands

The DH-HMAC-CHAP_Reply message and the DH-HMAC-CHAP_Success2 message are sent from the host to the controller, therefore they are mapped to the Authentication Send command. The DH-HMAC-CHAP_Challenge message and the DH-HMAC-CHAP_Success1 message are sent from the controller to the host, therefore they are mapped to the Authentication Receive command.