The Design and Formal Analysis of the Symmetric Key Distribution Protocol

John G. Underhill

Quantum Resistant Cryptographic Solutions Corporation
contact@qrcscorp.ca

December 1, 2025

Abstract

The Symmetric Key Distribution Protocol (SKDP) is a two-party symmetric authenticated key establishment protocol for deployments in which long-term symmetric derivation keys are provisioned before communication begins. The protocol establishes two directionally independent authenticated encryption channels by combining a hierarchical device-key structure, fresh random session tokens, SHA3 transcript hashes, cSHAKE-based derivation, KMAC-based authentication for the exchange messages, and authenticated encryption with the serialized packet header supplied as associated data. The implemented protocol is fully symmetric and does not rely on public-key encryption, signatures, certificates, or asymmetric key agreement.

This paper gives a formal and implementation-aligned security analysis of SKDP. The model captures the concrete reference behavior: a 21-byte packet header ordered as flag, message length, sequence number, and UTC timestamp; a six-message connect, exchange, and establish sequence; strict packet ordering; a 60-second packet-time acceptance window; AES-256-GCM as the default selected AEAD in the provided configuration; optional RCS operation when compiled for that mode; raw long-term key serialization; and the absence of in-band revocation, ratcheting, and complete keep-alive processing in the reviewed implementation. The analysis proves scoped authentication, session-key indistinguishability, replay resistance, and channel composition under explicit assumptions on random-token uniqueness, cSHAKE pseudorandomness, KMAC unforgeability, AEAD confidentiality and integrity, correct parser bounds enforcement, and secrecy of the long-term derivation keys.

The security results are intentionally bounded. SKDP provides session separation and fresh per-session traffic keys, but it does not provide full forward secrecy against later compromise of the relevant long-term device or server derivation key, because recorded exchange messages can be recomputed by an adversary who later obtains those derivation keys. The protocol also does not hide endpoint identities, does not provide anonymity, does not implement key revocation or at-rest key protection by itself, and relies on deployment policy for serialized-key confidentiality and integrity. These limitations are stated as part of the formal model rather than treated as external implementation concerns.

Contents

1 Introduction

1.1 Context and Motivation

Symmetric key distribution remains an important problem in systems where a public-key infrastructure is unavailable, undesirable, or operationally too expensive. Embedded devices, constrained terminals, payment devices, closed industrial networks, and application-specific infrastructure frequently begin with pre-provisioned symmetric keys. The technical problem is not simply to encrypt data with a shared key. A deployable protocol must authenticate both peers, derive fresh session keys, separate traffic directions, reject replayed or malformed packets, and limit the security impact of a single device-key disclosure.

SKDP addresses this design space by using a hierarchical symmetric key structure and a short two-party handshake. A master derivation key produces server derivation keys; a server derivation key produces device derivation keys; and the device and server use fresh session tokens to derive two directionally separated channel states. The protocol uses SHA3, cSHAKE, and KMAC as specified in the SHA-3 standard family [1, 2]. Application traffic is protected by the selected authenticated encryption function, with the packet header supplied as associated data. In the reviewed configuration, the selected AEAD wrapper resolves to AES-256-GCM unless the implementation is compiled with the RCS option enabled. AES and GCM are separately specified by NIST [3, 4]; the general AEAD interface used in modern protocol analysis is described in RFC 5116 [6].

The protocol is intended for post-quantum symmetric security in the narrow sense that its routine operation uses symmetric and hash-based primitives. This phrase does not imply protection against endpoint compromise, state exposure, or compromise of long-term symmetric derivation keys. In this paper, all such assumptions are made explicit.

1.2 Contributions

This work provides a formal analysis of SKDP as implemented, not of an idealized successor protocol. Its contributions are the following. First, it specifies the concrete message flow, key schedule, and packet-validation rules required for formal reasoning. Second, it defines a symmetric authenticated-key-establishment model for SKDP that includes active network control, packet replay, timestamp windows, parser bounds, and state transitions. Third, it provides scoped security definitions for authentication, session-key indistinguishability, replay resistance, and channel composition. Fourth, it gives proof

sketches reducing these properties to cSHAKE pseudorandomness, KMAC unforgeability, AEAD security, token freshness, and long-term key secrecy. Fifth, it identifies non-goals and limitations, including the absence of full forward secrecy, revocation, anonymity, at-rest key protection, and in-band ratcheting.

1.3 Organization of the Paper

Section 2 gives the engineering specification used by the proofs. Section 3 defines notation and the cryptographic model. Section 4 describes related work. Section 5 gives the formal protocol specification. Section 6 defines security properties. Section 7 presents the proof arguments. Section 8 discusses cryptanalytic and implementation considerations. Section 9 analyzes limitations and deployment requirements. Section 10 concludes.

2 Engineering Specification of SKDP

2.1 Protocol Roles and Trust Structure

SKDP has two online roles: a client C and a server S. The client is provisioned with a device derivation key DDK and a 16-byte key identity string. The server is provisioned with a server derivation key SDK and the server key identity. The master derivation key MDK is used to create server keys and is not part of the online session exchange.

The trust model is symmetric. There are no certificates, public keys, signatures, or third-party online validators in the implemented SKDP handshake. Authentication is based on the ability of the client to use its DDK, and the ability of the server to derive the same DDK from its SDK and the received device identity. The server therefore occupies a stronger trust position than an individual client device. Compromise of a device key affects that device. Compromise of the corresponding server derivation key can allow derivation of device keys within that server namespace.

2.2 Selected Primitives and Parameter Sets

SKDP uses the following primitives and implementation abstractions.

Primitive	Use in SKDP	Assumption in the model

SHA3	Session transcript hashes and verification-token hashing	Collision and preimage resistance in the modeled domain
cSHAKE	Long-term and session derivation	Pseudorandomness under secret keyed input and domain-separated customization
KMAC	Exchange-message authentication	Existential unforgeability under chosen-message attack
AES-256-GCM	Default selected AEAD in the reviewed configuration	AEAD confidentiality and integrity under nonce/state uniqueness and correct use
RCS	Optional selected AEAD when compiled with the RCS option	AEAD confidentiality and integrity as an implementation assumption
RNG	Long-term keys and session tokens	Uniformity and non-repetition of generated secret tokens

The 256-bit profile uses 32-byte derivation keys, tokens, and transcript hashes. In the default AES-GCM configuration, the AEAD authentication tag is 16 bytes. In the RCS configuration the tag length follows the RCS parameter set. The 512-bit profile is defined only with the RCS option in the current header logic and uses 64-byte key and tag fields.

2.3 Packet Header and Wire Format

Every SKDP packet begins with a 21-byte header. The implementation serializes it in the following exact order using little-endian integer encoding:

Offset	**Size**	**Field**
0	1 byte	packet flag
1	4 bytes	message length
5	8 bytes	sequence number
13	8 bytes	UTC packet creation time

The serialized header is not encrypted. For exchange request and exchange response messages, it is authenticated by KMAC together with the token cipher-

text. For establish and application data messages, it is supplied as AEAD associated data. Consequently, an attacker may observe flags, lengths, sequence numbers, and timestamps, but cannot alter authenticated headers without causing verification failure, except in phases where the receiver rejects the packet before authentication.

2.4 Packet Flags, Error Codes, and Message Types

The implemented packet flags are: none $0x00$, connect request $0x01$, connect response $0x02$, connection terminate $0x03$, encrypted message $0x04$, exchange request $0x05$, exchange response $0x06$, establish request $0x07$, establish response $0x08$, establish verify $0x09$, keep-alive request $0x0A$, session established $0x0B$, and error condition $0x0C$.
The implemented error codes include authentication failure, key-exchange authentication failure, channel down, invalid input, unrecognized key, random generation failure, network receive and transmit failure, unknown protocol, unsequenced packet, packet expiration, and general failure. The error code space is part of observable behavior. A peer can distinguish some rejection causes; this is a minor diagnostic oracle and should be considered in hardened deployments.

2.5 Long-Term Key Generation

The master key is sampled by the system random generator. Server and device keys are derived deterministically using cSHAKE. Abstractly,

$$\mathsf{SDK} = \mathsf{cSHAKE}(\mathsf{MDK}; \mathsf{CFG}, \mathsf{KID}_S),$$

$$\mathsf{DDK} = \mathsf{cSHAKE}(\mathsf{SDK}; \mathsf{CFG}, \mathsf{KID}_C).$$

The exact input ordering follows the implementation; the security model requires only that the cSHAKE call be domain-separated by the configuration string and identity bytes and that the secret derivation key remains unknown to the adversary for sessions claimed secure.
Serialized master, server, and device key records consist of the identity string, raw derivation key bytes, and expiration time. The core SKDP serialization routines do not authenticate or encrypt these records. Protection of serialized keys at rest and in transit is an operational requirement outside the cryptographic handshake.

2.6 Session Transcript Hashes

The connect request body is

$$\mathsf{CR} = \mathsf{KID}_C \parallel \mathsf{CFG}_C \parallel \mathsf{STOK}_C,$$

where STOK_C is generated by the client. The device session hash is

$$\mathsf{DSH} = H(\mathsf{CR}).$$

The connect response body is

$$\mathsf{CS} = \mathsf{KID}_S \parallel \mathsf{CFG}_S \parallel \mathsf{STOK}_S,$$

where STOK_S is generated by the server. The server session hash is

$$\mathsf{SSH} = H(\mathsf{CS}).$$

These hashes bind the visible identity, configuration, and connect-token fields into the later derivation steps.

2.7 Exchange Construction

The exchange messages do not use the selected AEAD. They use a pad-and-MAC construction derived from cSHAKE and KMAC. For a transcript hash $X \in \{\mathsf{DSH}, \mathsf{SSH}\}$ and device derivation key DDK, cSHAKE is used to generate a pad key stream and a MAC key. If M is the token to be transported, the sender computes

$$C = M \oplus \mathsf{Pad}(\mathsf{DDK}, X),$$

$$T = \mathsf{KMAC}_{K_M}(C \parallel \langle \mathsf{Header} \rangle).$$

The receiver recomputes T and accepts only on a constant-time tag match. If verification succeeds, the receiver recovers M by XORing the same pad stream. The client exchange request transports the device token key DTK. The server exchange response transports the server token key STK. These transported values are random session secrets and are distinct from the visible connect tokens STOK_C and STOK_S.

2.8 Channel-Key Derivation

The client-to-server channel is derived from DTK and DSH. The server-to-client channel is derived from STK and SSH. Abstractly,

$$(K_{C\to S}, N_{C\to S}) = mathsf cSHAKE(\mathsf{DTK}; \mathsf{DSH}),$$

$$(K_{S\to C}, N_{S\to C}) = mathsf{cSHAKE}(\mathsf{STK}; \mathsf{SSH}).$$

The implementation squeezes enough bytes from cSHAKE to initialize the selected cipher key and nonce fields. The key and nonce are therefore co-derived from the same token and transcript hash. This is safe in the model because the token is fresh and secret for each session, but the co-derivation is a protocol requirement and not a property supplied by the AEAD alone.

2.9 Establish Phase and Key Confirmation

After both exchange tokens have been recovered and both directional cipher states have been initialized, the client generates a verification token VTOK. It encrypts this token under the client-to-server channel with the establish-request header as associated data. The server decrypts the message, hashes the recovered token, encrypts the hash under the server-to-client channel with the establish-response header as associated data, and sends the result to the client. The client decrypts the response and compares it with its own hash of VTOK. A successful comparison provides explicit confirmation to the client that the server received VTOK under the client-to-server channel and replied under the server-to-client channel. The server obtains implicit assurance from successful decryption of the establish request; the implemented handshake does not add a final server-side confirmation message after the client comparison.

2.10 Application Data Processing

For application data, the sender requires an established session state, checks that the plaintext length does not exceed `SKDP_MESSAGE_SIZE`, increments the transmit sequence number, sets the encrypted-message flag, writes the UTC timestamp, serializes the header, supplies that header as associated data, and invokes the selected AEAD transform. The receiver requires the encrypted-message flag, the expected next sequence number, a valid timestamp, a ciphertext length bounded by `SKDP_MESSAGE_SIZE + SKDP_MACTAG_SIZE`, and sufficient output-buffer capacity. Only after successful authentication does the receiver advance the receive sequence. On authentication failure, the implementation clears the exchange flag and the session is no longer usable.
This ordering is a security boundary. A construction with stateful AEAD processing must not advance the receive counter or retain authenticated-encryption state after rejecting a forged packet.

2.11 Keep-Alive, Revocation, Ratcheting, and Storage Scope

The reviewed code contains a server-side keep-alive send helper and a keep-alive state structure, but it does not implement a complete interoperable keep-alive request/response state machine. The protocol analysis therefore treats keep-alive as non-normative.

The reviewed core protocol does not implement revocation lists, online status checks, authenticated key containers, encrypted key storage, or in-band ratcheting. These features may be added by deployment policy or future protocol extensions, but they are not security properties of the analyzed construction.

3 Cryptographic Model and Assumptions

3.1 System Model

A session is an execution of the six-message SKDP handshake between a client instance and a server instance. Each instance has local state containing the long-term material available to that role, identity strings, configuration strings, transcript hashes, token values, transmit and receive counters, packet timestamp checks, selected AEAD states, and an exchange flag. A session accepts when its exchange flag reaches the established state and both directional channel states are initialized.

A peer is partnered with an accepting session if it has the same connect transcript, the same exchange transcript, matching derived directional keys in opposite roles, and the complementary role. This is a standard partnering relation adapted to a symmetric authenticated-key-establishment setting [7, 8].

3.2 Adversarial Model

The adversary controls the network. It may observe, delay, replay, drop, reorder, modify, and inject packets. It sees headers, message lengths, timestamps, identities, configuration strings, and encrypted payloads. It may start concurrent sessions and may cause honest endpoints to process malformed inputs. It may obtain chosen-message outputs from honest protocol oracles by causing them to execute their prescribed algorithms.

The adversary does not break the implementation of SHA3, cSHAKE, KMAC, or the selected AEAD except through the standard games assigned to those primitives. It does not control the random generator of honest endpoints. It does not control the local clocks of honest endpoints, although it may exploit clock skew within the configured acceptance window.

Long-term key compromise is modeled explicitly. A session is *fresh for long-term-key security* only if the adversary has not obtained the relevant DDK, SDK, or MDK before or after the session when past-session secrecy is claimed. If the adversary later obtains a long-term derivation key and has recorded the exchange messages, SKDP does not provide full forward secrecy for that session. This restriction is part of the definition, not a proof artifact.

3.3 Cryptographic Assumptions

Assumption 3.1 (Random-token uniqueness)**.** *All honest values* STOK_C, STOK_S, DTK, STK, *and* VTOK *are sampled uniformly from their configured byte domains and repeat only with negligible probability.*

Assumption 3.2 (cSHAKE pseudorandomness)**.** *cSHAKE, when keyed by a secret derivation key or secret session token and invoked with the protocol's customization inputs, is computationally indistinguishable from a pseudorandom function to an adversary that does not know the secret input.*

Assumption 3.3 (KMAC unforgeability)**.** *KMAC is existentially unforgeable under chosen-message attack for the configured key and tag lengths.*

Assumption 3.4 (AEAD security)**.** *The selected AEAD provides confidentiality and ciphertext integrity for the way it is invoked by SKDP: distinct session-derived key and nonce material per direction, serialized headers as associated data, and no acceptance of unauthenticated ciphertext.*

Assumption 3.5 (Parser and state enforcement)**.** *Implementations enforce the message-length, header-length, output-capacity, flag, sequence, and timestamp checks specified in Section 2. Packets failing these checks are rejected without exposing plaintext and without advancing the receive sequence.*

3.4 Post-Quantum Interpretation

SKDP's routine security assumptions are symmetric and hash based. Grover-type quantum search reduces the effective brute-force margin of an n-bit symmetric key to roughly $2^{n/2}$ oracle evaluations in the idealized model. The 256-bit symmetric profile is therefore conventionally associated with approximately 128-bit post-quantum brute-force cost, while the 512-bit profile raises that margin. This statement does not imply protection against endpoint compromise, random-generator failure, side channels, or long-term derivation-key disclosure.

4 Related Work

4.1 Authenticated Key Exchange Models

The formal treatment of entity authentication and authenticated key distribution begins with the Bellare-Rogaway model [7]. Later compositional models, including the Canetti-Krawczyk framework, connect key-exchange security to secure-channel construction [8]. SKDP is not a Diffie-Hellman protocol and does not fit the common public-key authenticated-key-exchange setting directly. Nevertheless, the same concepts of session partnering, key indistinguishability, explicit authentication, and channel composition are appropriate once the pre-provisioned symmetric derivation key is treated as the authentication credential.

4.2 Symmetric Key Distribution Systems

Classic systems such as Needham-Schroeder and Kerberos use symmetric keys to distribute session material through a trusted server [9, 10]. Telecommunications authentication and key-agreement mechanisms also rely on long-term symmetric roots, sequence numbers, random challenges, and MAC-based validation. SKDP differs by operating as a two-party protocol in which the server derives a device key locally rather than consulting an online ticket authority. DUKPT-style payment systems use hierarchical derivation and transaction-specific keys. SKDP shares the goal of limiting direct reuse of long-term keys, but it includes a two-party authenticated handshake, transcript binding, and a duplex encrypted channel rather than deriving isolated transaction keys only.

4.3 AEAD and Pre-Shared-Key Secure Channels

Modern secure channels use AEAD constructions to protect both ciphertext and associated metadata [6]. TLS pre-shared-key modes similarly combine transcript binding with symmetric key schedules, although TLS commonly combines PSK material with asymmetric or ephemeral key agreement in stronger modes. SKDP should be understood as a standalone symmetric-only secure-channel establishment mechanism. Its advantages are simplicity and primitive minimalism. Its principal trade-off is the absence of forward secrecy against later disclosure of long-term symmetric derivation keys.

5 Formal Protocol Specification

5.1 Notation

Let C denote the client and S the server. Let KID_C and KID_S denote identity strings, and let CFG denote the selected protocol configuration string. Let H denote SHA3, F denote cSHAKE as a keyed pseudorandom function, and M denote KMAC. Let $\mathcal{E}.\mathsf{Enc}$ and $\mathcal{E}.\mathsf{Dec}$ denote the selected AEAD encryption and decryption functions. Let $\langle h \rangle$ denote the serialized SKDP packet header for the message under consideration.

5.2 Connect Messages

The client samples STOK_C and sends

$$C \to S : \mathsf{CR} = \mathsf{KID}_C \parallel \mathsf{CFG}_C \parallel \mathsf{STOK}_C.$$

Both sides compute

$$\mathsf{DSH} = H(\mathsf{CR}).$$

The server validates the recognized identity prefix, derives or selects the relevant device key, compares the configuration string, samples STOK_S, and sends

$$S \to C : \mathsf{CS} = \mathsf{KID}_S \parallel \mathsf{CFG}_S \parallel \mathsf{STOK}_S.$$

Both sides compute

$$\mathsf{SSH} = H(\mathsf{CS}).$$

5.3 Exchange Messages

For the exchange request the client samples DTK and derives

$$(K_1^p, K_1^m) = F(\mathsf{DDK}; \mathsf{DSH}).$$

It computes

$$X_1 = \mathsf{DTK} \oplus \mathsf{Pad}(K_1^p), \qquad T_1 = M_{K_1^m}(X_1 \parallel \langle h_1 \rangle),$$

and sends (X_1, T_1). The server recomputes (K_1^p, K_1^m), verifies T_1, recovers DTK, and derives

$$(K_{C\to S}, N_{C\to S}) = F(\mathsf{DTK}; \mathsf{DSH}).$$

For the exchange response the server samples STK, derives

$$(K_2^p, K_2^m) = F(\mathsf{DDK}; \mathsf{SSH}),$$

computes

$$X_2 = \mathsf{STK} \oplus \mathsf{Pad}(K_2^p), \qquad T_2 = M_{K_2^m}(X_2 \parallel \langle h_2 \rangle),$$

and sends (X_2, T_2). The client verifies T_2, recovers STK, and derives

$$(K_{S\to C}, N_{S\to C}) = F(\mathsf{STK}; \mathsf{SSH}).$$

5.4 Establish Messages

The client samples VTOK and sends

$$C \to S : Y_1 = \mathcal{E}.\mathsf{Enc}_{K_{C\to S}, N_{C\to S}}(\langle h_3 \rangle, \mathsf{VTOK}).$$

The server decrypts Y_1, computes $H(\mathsf{VTOK})$, and sends

$$S \to C : Y_2 = \mathcal{E}.\mathsf{Enc}_{K_{S\to C}, N_{S\to C}}(\langle h_4 \rangle, H(\mathsf{VTOK})).$$

The client decrypts Y_2 and accepts if the recovered value equals its local $H(\mathsf{VTOK})$. The server sets its established state after producing the establish response. The client sets its established state after verifying the response.

5.5 Application Data

An application packet in direction $d \in \{C \to S, S \to C\}$ is

$$Z = \mathcal{E}.\mathsf{Enc}_{K_d, N_d}(\langle h \rangle, P),$$

where P is the plaintext and $\langle h \rangle$ includes the encrypted-message flag, ciphertext length including tag, next sequence number, and UTC time. A receiver accepts only if the flag is correct, the sequence is exactly the next expected value, the timestamp lies within the acceptance window, the length is within bounds, the output buffer is sufficient, and AEAD verification succeeds.

6 Security Definitions

6.1 Session Correctness

Definition 6.1 (Correctness)**.** *An honest SKDP client and honest SKDP server executing the protocol with corresponding keys, matching configuration strings, valid clocks, non-repeating random tokens, and reliable delivery accept with matching directional keys: the client's transmit state equals the server's receive state for the client-to-server direction, and the server's transmit state equals the client's receive state for the server-to-client direction.*

6.2 Authentication

Definition 6.2 (Peer authentication)**.** *A client session authenticates a server if client acceptance implies the existence of a partnered server session that processed the same connect and exchange transcript and generated the establish response under the derived server-to-client channel. A server session authenticates a client if server establishment implies that a party possessing the corresponding device derivation key generated a valid exchange request and establish request for the same transcript.*

This definition distinguishes explicit and implicit authentication. The client receives explicit key confirmation through the verification-token hash. The server receives implicit confirmation through successful verification of the exchange request and successful AEAD decryption of the establish request.

6.3 Key Indistinguishability

Definition 6.3 (Session-key indistinguishability)**.** *For a fresh session in which the relevant long-term derivation keys remain secret, an adversary who observes the full transcript and controls the network cannot distinguish either accepted channel key from a uniformly random key of the same length with advantage greater than the sum of the assumed advantages against cSHAKE pseudorandomness, KMAC unforgeability, AEAD security, and random-token uniqueness.*

This definition excludes sessions for which the adversary later obtains the relevant DDK or SDK and has recorded the token ciphertexts. That exclusion reflects the protocol's symmetric-only key schedule.

6.4 Replay Resistance

Definition 6.4 (Replay resistance)**.** *A replayed or reordered packet is rejected except with negligible probability if the implementation enforces strict sequence equality, timestamp validity where required, header authentication, and nonmutating rejection of failed authenticated packets.*

6.5 Channel Security

Definition 6.5 (Application channel security)**.** *After acceptance, each direction of the SKDP channel is secure if application packets in that direction satisfy AEAD confidentiality and integrity with the serialized SKDP header as associated data and with the receiver accepting each sequence number at most once.*

7 Security Analysis

7.1 Correctness

Theorem 7.1 (Correctness)**.** *Under honest execution and successful random generation, the client and server derive matching directional channel states and the client accepts the establish response.*

Proof. Both parties compute $\mathsf{DSH} = H(\mathsf{CR})$ and $\mathsf{SSH} = H(\mathsf{CS})$ from identical byte strings. The server derives the same DDK that was provisioned to the client by applying the deterministic device-key derivation to the same identity and configuration inputs. The client encrypts DTK under a cSHAKE pad derived from DDK and DSH; the server derives the same pad and recovers DTK. Both derive $(K_{C\to S}, N_{C\to S})$ from DTK and DSH. The same argument applies to STK and SSH in the reverse direction. The establish request and response are then encrypted under complementary AEAD states. The client accepts exactly when the decrypted response equals $H(\mathsf{VTOK})$, which is the value the honest server computed from the decrypted establish request. □

7.2 Exchange Authentication

Lemma 7.1 (Exchange-message unforgeability)**.** *Assuming KMAC is unforgeable and cSHAKE-derived MAC keys are pseudorandom, an adversary that does not know the relevant* DDK *cannot produce a new accepted exchange request or exchange response for a fresh transcript except with negligible probability.*

Proof. For a fixed transcript hash and header, acceptance of an exchange message requires a valid KMAC tag over $X \parallel \langle h \rangle$ under a MAC key derived from DDK and the transcript hash. If the derived MAC key is pseudorandom to the adversary, any new accepted pair (X, T) gives a KMAC forgery. Modifying any header field changes $\langle h \rangle$ and therefore also requires a new valid tag. Replaying an old exchange message into a different transcript changes the transcript hash and therefore changes the derived MAC key. □

7.3 Token Confidentiality

Lemma 7.2 (Exchange-token confidentiality)**.** *Assuming cSHAKE-derived pad bytes are pseudorandom and not reused for the same secret transcript, the exchange ciphertexts reveal no computational information about* DTK *or* STK *to an adversary lacking* DDK*.*

Proof. Each exchange token is XORed with a pad derived from DDK and a fresh transcript hash containing a fresh connect token. Under the cSHAKE pseudorandomness assumption, the pad is indistinguishable from a uniform byte string of the token length. XOR with such a pad is computationally indistinguishable from a one-time pad. The transcript freshness assumption prevents pad reuse except with negligible probability. □

7.4 Session-Key Indistinguishability

Theorem 7.2 (Session-key indistinguishability for fresh sessions)**.** *For a fresh SKDP session in which the relevant long-term derivation keys remain secret, the accepted directional channel keys are computationally indistinguishable from uniformly random keys to a network adversary.*

Proof. By Lemma 6.3, DTK and STK are computationally hidden from any adversary lacking DDK. These tokens are sampled uniformly and independently. The directional channel keys are cSHAKE outputs keyed by DTK and STK with transcript hashes as customization input. Under cSHAKE pseudorandomness, the resulting key and nonce material is indistinguishable from random. The adversary's view consists of public transcript fields, authenticated token ciphertexts, and AEAD ciphertexts. None of these reveal the session tokens except through breaking the assumed primitives. □

7.5 Authentication and Key Confirmation

Theorem 7.3 (Scoped authentication)**.** *If a client accepts an SKDP session, then except with negligible probability there exists a server instance that derived the same server-to-client channel and encrypted the hash of the client's verification token. If a server reaches its established state, then except with negligible probability the establish request was generated by a party that recovered the client-to-server channel key.*

Proof. Client acceptance requires successful AEAD decryption of the establish response and equality with $H(\mathsf{VTOK})$. The only party that can compute this plaintext and authenticate it under the server-to-client channel is a party that recovered STK, derived the channel key, and obtained VTOK by decrypting the establish request. Forging the response requires either an AEAD forgery, a hash collision/preimage success against the verification-token hash relation, or recovery of the channel key. Server establishment follows from successful verification of the client exchange request and successful AEAD decryption of the establish request. This gives implicit server-side assurance. It is not a final explicit confirmation that the client has decrypted the establish response. □

7.6 Replay and Reordering Resistance

Theorem 7.4 (Replay resistance under state enforcement)**.** *Assuming timestamp validation, strict sequence equality, header authentication, and non-mutating rejection of failed authenticated packets, accepted SKDP packets cannot be replayed or reordered within an honest session except with negligible probability.*

Proof. Exchange packets carry timestamps and headers authenticated by KMAC. A replay outside the timestamp window is rejected before acceptance. A replay inside the window must also match the expected sequence and state; otherwise it is rejected. Establish and application packets bind the serialized header as AEAD associated data. Changing a sequence number, timestamp, flag, or length therefore requires an AEAD forgery. Replaying an already accepted packet presents an old sequence number and is rejected. Reordering a future packet presents a sequence number different from the next expected value and is rejected. If a forged in-sequence packet fails AEAD verification, the implementation must not advance receive state; otherwise the adversary can cause denial of service. The revised state rule in Section 2 is therefore part of the theorem's hypothesis. □

7.7 AEAD Channel Composition

Theorem 7.5 (Secure-channel composition)**.** *If an SKDP session accepts with indistinguishable directional channel keys and the selected AEAD satisfies confidentiality and integrity for the SKDP invocation pattern, then application data in each direction satisfies confidentiality and ciphertext integrity with respect to the serialized packet header as associated data.*

Proof. By Theorem 6.4, the channel keys are indistinguishable from random for fresh sessions. Each application packet is processed by the selected AEAD under the corresponding directional key, with a unique authenticated header containing the sequence number and timestamp. Confidentiality follows from AEAD confidentiality. Integrity of both plaintext and associated header fields follows from AEAD ciphertext integrity. Directional separation prevents a ciphertext from one direction from being accepted in the other direction because the receive key differs from the transmit key for the opposite role. □

7.8 Forward-Secrecy Boundary

Theorem 7.6 (No full forward secrecy against long-term derivation-key compromise)**.** *SKDP does not provide full forward secrecy if the adversary records*

a completed exchange and later obtains the corresponding DDK *or a derivation key sufficient to recompute it.*

Proof. Given DDK, the recorded connect messages, and the recorded exchange headers, the adversary recomputes DSH, SSH, the cSHAKE pads, and the KMAC keys used in both exchange messages. It then decrypts DTK and STK from the recorded XOR-protected token ciphertexts. From these tokens and the transcript hashes it recomputes both directional channel keys. Therefore the protocol cannot satisfy a standard forward-secrecy definition with respect to later compromise of the long-term derivation key. □

The protocol still provides per-session uniqueness: compromise of one accepted session's traffic keys does not by itself reveal a different session's traffic keys when the random tokens are independent and the long-term derivation keys remain secret.

8 Cryptanalytic and Implementation Considerations

8.1 Malformed Packet Handling

Malformed inputs are a cryptographic boundary in SKDP because packet lengths are attacker-controlled until parsed and authenticated. A conforming implementation shall reject streams shorter than the header length before deserializing the header, shall verify that message length does not exceed the stream body length, shall verify destination buffer capacity before copying, and shall reject encrypted data packets whose body length is smaller than the authentication tag or larger than the maximum permitted message plus tag. Establish packets shall have exact expected lengths. These checks are not optional hardening; without them, the security model fails before the cryptographic assumptions are reached.

8.2 State Mutation on Failed Authentication

The selected AEAD wrapper may contain stateful processing. For this reason, failed authentication must place the session into a terminal or teardown-required state, and the receive sequence number must advance only after successful authentication. If an unauthenticated in-sequence packet advances the sequence number or leaves the cipher in a partially updated state, an attacker can desynchronize the session with a single forged packet. The formal replay theorem assumes non-mutating rejection of forged packets.

8.3 Timestamp Failure Modes

Timestamps provide a bounded freshness check but do not replace cryptographic authentication. A clock set too far in the past or future can cause denial of service by rejecting valid packets. An adversary able to shift clocks may enlarge the effective replay window. Strict sequence numbers and authenticated headers limit this effect for established traffic, but the connect and exchange phases still rely partly on the configured time threshold. Deployments requiring operation without reliable UTC should add an explicit nonce or epoch mechanism and analyze it as part of the transcript.

8.4 Metadata Exposure

SKDP does not encrypt packet headers. The adversary observes packet type, body length, sequence number, timestamp, and visible identities in the connect messages. The protocol therefore provides no anonymity, unlinkability, traffic-flow confidentiality, or hiding of client/server identities. Its confidentiality claims apply to transported token secrets and application plaintext after session establishment.

8.5 Key Storage and Provisioning

The cryptographic handshake assumes correct provisioning and secrecy of MDK, SDK, and DDK. The serialization functions for these keys do not add encryption or authentication. A conforming deployment that stores serialized keys shall protect them using an external authenticated and confidential storage mechanism, such as a hardware security module, OS-protected keystore, or application-level authenticated-encryption container. Without such protection, key-file tampering can alter expiration or substitute secret material, and key-file disclosure compromises the corresponding trust scope.

8.6 Revocation, Ratcheting, and Keep-Alive

The formal model does not include key revocation, in-band ratcheting, or a complete keep-alive protocol. A deployment may expire keys by time and may reject unknown identities, but there is no intrinsic revocation list or online status check in the core SKDP exchange. Similarly, there is no implemented rekey exchange that injects new entropy into an established session. Adding these functions would require new messages, new state transitions, and new proofs.

8.7 Side Channels

The protocol uses constant-time comparison for tags and fixed-length token comparisons in the reference code, but this paper does not prove constant-time execution of the full implementation. Packet parsing, timestamp rejection, error-code selection, and network behavior may leak coarse rejection causes. Implementations in adversarial physical environments should use hardened key storage, constant-time cryptographic primitives, compiler-aware zeroization, and uniform error handling where diagnostic exposure is unacceptable.

8.8 Denial of Service

SKDP performs low-cost parsing and state checks before expensive cryptographic work where possible. Nevertheless, any unauthenticated server-facing protocol is susceptible to connection floods, incomplete handshakes, and repeated malformed packets. The reviewed server interface is a blocking single-session primitive rather than a complete high-scale service. Production deployment requires external accept-loop policy, socket timeouts, concurrency limits, and rate limiting. These mechanisms are operational controls rather than cryptographic properties.

9 Parameter and Conformance Analysis

9.1 Operational Limits

The implementation defines a maximum plaintext size of 1024 bytes for application messages. Encrypted packets include the selected AEAD tag in the message length. Packet times are accepted only within a 60-second threshold around local UTC. The sequence terminator constant is a 32-bit value, while runtime sequence counters are 64-bit values. A conservative implementation should close or rekey long before sequence exhaustion or collision with special control values.

9.2 Conformance Requirements

An implementation conforms to this analysis only if it satisfies the following requirements. It shall serialize headers in the exact 21-byte order specified in Section 2. It shall authenticate the serialized header in every exchange, establish, and application packet requiring authentication. It shall reject data packets that do not carry the encrypted-message flag. It shall reject oversized

and undersized ciphertexts before decryption. It shall pass an explicit output-buffer capacity into packet parsing and decryption. It shall advance receive counters only after successful authentication. It shall treat authentication failure as teardown-required for the affected session. It shall use independent directional channel states. It shall protect serialized long-term keys outside the core serialization format.

9.3 Test and Validation Expectations

The minimum validation set for SKDP should include deterministic key-derivation vectors, successful client/server handshake tests, wrong-key and wrong-configuration rejection tests, exchange-MAC tamper tests, establish-message length tests, oversized packet tests, stream-parser truncation tests, header tamper tests, sequence replay tests, timestamp-expiration tests, corrupted AEAD tag tests, and tests proving that a forged in-sequence packet does not advance the receive sequence. Memory-safety validation should include address and undefined-behavior sanitizers for parser and decrypt paths.

10 Limitations and Deployment Requirements

10.1 Security Limitations

The following are inherent limitations of the analyzed construction.

1. SKDP does not provide full forward secrecy against later compromise of DDK, SDK, or MDK.
2. SKDP does not provide anonymity or identity hiding because connect messages carry visible identity strings.
3. SKDP does not implement revocation or online status checking.
4. SKDP does not protect serialized keys at rest by itself.
5. SKDP relies on random-token uniqueness and correct random generation.
6. SKDP relies on clock validity for bounded replay filtering.
7. SKDP relies on correct parser bounds enforcement before cryptographic processing.

10.2 Deployment Requirements

A secure deployment shall keep master and server derivation keys in strongly protected administrative environments. Device keys should be provisioned through an authenticated channel or in a protected manufacturing process. Serialized key records should be encrypted and authenticated by a storage-layer mechanism. Server implementations should add rate limiting, socket timeouts, connection quotas, and logging policies appropriate to the deployment. If revocation is required, the deployment must add an authenticated revocation or allow-list mechanism before accepting a device identity.

10.3 Future Work

Future extensions may add a formally specified ratchet, a revocation mechanism, encrypted key containers, reliable keep-alive semantics, and a non-time-based replay mechanism for clockless environments. Any such extension should introduce explicit message flags, transcript labels, state transitions, and test vectors. Forward secrecy requires an additional ephemeral secret contribution that cannot be recomputed from recorded traffic and later long-term key disclosure; this may be supplied by a post-quantum KEM, an out-of-band one-time symmetric secret, or another independently analyzed mechanism.

11 Conclusion

SKDP is a compact symmetric authenticated key establishment protocol whose implemented design can be analyzed cleanly when its assumptions are stated precisely. The six-message handshake derives two independent channel states from long-term symmetric derivation keys, fresh random token keys, and transcript hashes. Exchange messages are protected by cSHAKE-derived pads and KMAC tags over ciphertext and serialized headers. Establish and application messages use the selected AEAD with the serialized header as associated data. Under the assumptions stated in this paper, SKDP provides scoped peer authentication, session-key indistinguishability, replay resistance, and authenticated channel composition for sessions whose long-term derivation keys remain secret.

The protocol's principal limitations are also clear. SKDP is not forward-secret against later compromise of the relevant long-term derivation key; it does not hide identities or metadata; it does not implement revocation, ratcheting, or at-rest key protection; and its security depends on correct parser and state-machine enforcement. These boundaries do not invalidate the protocol as a symmetric key distribution design. They define the conditions under which

its security claims are valid and the deployment controls required for practical use.

References

[1] National Institute of Standards and Technology, *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, August 2015. https://csrc.nist.gov/pubs/fips/202/final

[2] J. Kelsey, S. Chang, and R. Perlner, *NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash*, National Institute of Standards and Technology, December 2016. https://csrc.nist.gov/pubs/sp/800/185/final

[3] National Institute of Standards and Technology, *FIPS 197: Advanced Encryption Standard (AES)*, updated 2023. https://csrc.nist.gov/pubs/fips/197/final

[4] M. Dworkin, *NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, National Institute of Standards and Technology, November 2007. https://csrc.nist.gov/pubs/sp/800/38/d/final

[5] L. Chen, *NIST SP 800-108 Revision 1: Recommendation for Key Derivation Using Pseudorandom Functions*, National Institute of Standards and Technology, 2022. https://csrc.nist.gov/pubs/sp/800/108/r1/upd1/final

[6] D. McGrew, *RFC 5116: An Interface and Algorithms for Authenticated Encryption*, Internet Engineering Task Force, January 2008. https://datatracker.ietf.org/doc/html/rfc5116

[7] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution," *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science 773, Springer, 1994. https://link.springer.com/chapter/10.1007/3-540-48329-2_21

[8] R. Canetti and H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels," *Advances in Cryptology – EUROCRYPT 2001*, Lecture Notes in Computer Science 2045, Springer, 2001. https://eprint.iacr.org/2001/040

[9] R. Needham and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, vol. 21, no. 12, 1978.

[10] J. Kohl and C. Neuman, *RFC 1510: The Kerberos Network Authentication Service (V5)*, Internet Engineering Task Force, September 1993. https://datatracker.ietf.org/doc/html/rfc1510

[11] J. G. Underhill, *Symmetric Key Distribution Protocol – SKDP Technical Specification*, Quantum Resistant Cryptographic Solutions Corporation, 2024.

[12] Quantum Resistant Cryptographic Solutions Corporation, *SKDP Reference Source Code*, QRCS source distribution.

[13] Quantum Resistant Cryptographic Solutions Corporation, *QSC Cryptographic Library*, QRCS source distribution.