The Design and Formal Analysis of the Symmetric Key Distribution Protocol

John G. Underhill

Quantum Resistant Cryptographic Solutions Corporation

Abstract. The Symmetric Key Distribution Protocol (SKDP) is a lightweight authenticated key establishment mechanism designed for systems that rely on long term symmetric secrets and require post-quantum readiness without the use of public key cryptography. The protocol provides a compact three stage handshake that incorporates nonce based freshness, transcript binding, explicit key confirmation, and authenticated encryption of all session material. SKDP is intended for environments that include embedded devices, constrained networks, and symmetric only trust infrastructures where low computational cost and formal verifiability are primary requirements.
This paper presents the design and formal analysis of SKDP. The protocol is described in an engineering accurate manner based on the reference implementation, including exact message formats, header structure, and deterministic key derivation. A formal model is developed that captures the security goals of authenticated key exchange under a powerful network adversary. The analysis proves that SKDP achieves confidentiality, authenticity, replay protection, and explicit key confirmation when instantiated with any secure AEAD algorithm and a collision resistant hash function. The paper also compares SKDP with established symmetric key distribution mechanisms such as Kerberos, TLS pre shared key operation, and the mobile network AKA family, and positions it as a hash based alternative suitable for post-quantum symmetric environments. The results show that SKDP provides a formally verified and implementation aligned approach to symmetric authenticated key establishment with well defined security guarantees and practical deployment applicability.

1 Introduction

1.1 Context and Motivation

Symmetric key distribution remains a central problem in applied cryptography, particularly in systems that operate without public key infrastructure or where asymmetric computation is impractical. Examples include embedded deployments, large scale device networks, constrained IoT environments, and security architectures that favor symmetric trust anchors for long term post quantum resilience. In these settings, long term symmetric secrets must be used to authenticate endpoints and derive fresh session keys while maintaining resistance to replay, compromise, and future key exposure.

The Symmetric Key Distribution Protocol (SKDP) is designed precisely for such environments. SKDP assumes that the client and server each possess provisioned symmetric key material arranged in a hierarchy: a master derivation key, server keys derived from the master key, and device keys derived from server keys. Using only these long term symmetric keys and fresh randomness generated during the handshake, SKDP establishes an encrypted and authenticated duplexed channel.

All key derivation, hashing, and MAC computations in SKDP use the Keccak family of functions, specifically SHA3, cSHAKE, and KMAC, exactly as in the reference implementation. During the handshake, cSHAKE and KMAC are used to derive per session material, authenticate message fields, and bind packet headers. After the handshake completes, the channel is encrypted using the SKDP cipher wrapper, which instantiates an AEAD construction (RCS in the default build, or AES GCM when configured) and always treats the serialized packet header as associated data. Together, these mechanisms provide session key freshness, explicit key confirmation, and strong replay resistance without requiring any form of public key cryptography.

The motivation for SKDP is twofold. First, many deployments require low latency symmetric key establishment that can be implemented on devices lacking hardware acceleration for asymmetric operations. Second, long term migration strategies toward post quantum security increasingly rely on symmetric primitives and hash based derivation. SKDP offers a compact, fully symmetric handshake that integrates these primitives into a communication protocol with analyzable security properties.

1.2 Contributions

This work presents a precise and implementation aligned formal analysis of the Symmetric Key Distribution Protocol. The contributions are as follows.

- A complete description of the SKDP handshake based directly on the reference implementation, including message formats, packet header structure, the use of cSHAKE and KMAC for derivation and authentication, and the SKDP cipher based AEAD data channel (RCS by default, with an AES GCM option).

- A protocol model that captures the authenticated key establishment properties provided by SKDP, including assumptions on time synchronization, session token uniqueness, and adversarial control of the network.

- Formal security definitions suitable for symmetric authenticated key exchange, covering confidentiality, authenticity, replay resistance, forward secrecy, and explicit key confirmation.

- A reduction style analysis demonstrating that SKDP achieves its goals under standard assumptions about Keccak based primitives and the integrity of the SKDP AEAD construction.

- A detailed consistency analysis comparing the engineering implementation with the formal specification to ensure that the reference code realizes the protocol exactly as modeled.
- A contextual comparison situating SKDP among existing symmetric key distribution mechanisms, including DUKPT style hierarchical derivation schemes, key ladder based protocols, and symmetric only authenticated channel designs.

1.3 Organization of the Paper

Section 2 provides an engineering level description of SKDP, including message layouts, the packet header format, and the handshake sequence implemented in the code. Section 3 introduces the protocol model and enumerates all cryptographic and operational assumptions. Section 4 reviews related symmetric key establishment designs. Section 5 presents the formal protocol specification. Section 6 defines the security goals of SKDP. Section 7 contains the provable security analysis. Section 8 provides the cryptanalytic evaluation of the protocol and its components. Section 9 compares the formal specification with the reference implementation. Section 10 discusses performance and practical deployment considerations. Section 11 describes limitations and directions for future work, and Section 12 concludes the paper.

2 Engineering Specification of SKDP

This section provides a precise engineering level description of SKDP based entirely on the reference implementation. All message formats, packet fields, key derivations, hashing steps, and state transitions correspond exactly to the behaviour of the code in `skdp.c`, `skdpclient.c`, and `skdpserver.c`. The purpose of this section is to anchor the formal model in the actual packet layouts, transcript binding rules, and deterministic derivation logic realized by SKDP.

2.1 Packet and State Formats

All SKDP communication uses a fixed format packet structure. Each packet consists of a serialized header followed by a message body. The header is always transmitted in the clear. During the exchange, establish, and data phases, the serialized header is bound into the authentication layer as associated data. Both client and server enforce strict state transition rules based on packet sequence numbers, timestamp validity, and cryptographic authentication results.
Each endpoint maintains a session state that includes:

- separate transmit and receive sequence counters (`txseq`, `rxseq`),
- a pair of cipher states (`txcpr`, `rxcpr`) that are initialized during the handshake and used for all encrypted traffic,
- the transcript hash values `dsh` and `ssh` derived during the connect stage,
- the device or server derivation key used to produce handshake specific subkeys,
- an exchange state flag that records the current stage of the handshake.

2.1.1 Header Structure

Each packet begins with a 21 byte header. The layout is determined by the routines `skdp_packet_header_serialize` and `skdp_packet_header_deserialize`. All multi byte fields are little endian. The fields are:

Offset	Size	Description
0	1	packet flag (message type)
1	4	payload length in bytes
5	8	packet sequence number
13	8	UTC timestamp (seconds since epoch)

The header is treated as immutable associated data during authenticated encryption. Before encryption or decryption, the implementation serializes the header into a temporary buffer and passes it into the cipher state using `skdp_cipher_set_associated`. Any modification of header fields therefore causes authentication failure.

2.1.2 Packet Body

The body of the packet is defined by the message type. During the handshake, the message body contains identity fields, configuration strings, token bundles, encrypted key material, or hashed verification data depending on the phase. During normal data transmission it contains ciphertext followed by an authentication tag the length of which is fixed by the selected AEAD mode.

2.2 State Transition Rules

The protocol enforces strict ordering through sequence numbers and freshness through timestamp validation. For each received packet:

- The receiver checks `packet.sequence == ctx->rxseq`. Out of order packets are rejected and cause termination of the exchange.
- After acceptance, `rxseq` is incremented.
- Timestamp validation uses `skdp_packet_time_valid`, which accepts packets whose timestamps fall within a fixed window around the local clock. Packets outside this window are rejected with a packet expiry error.

For outgoing packets:

- The sender sets `packet.sequence = ctx->txseq`,
- calls `skdp_packet_set_utc_time` to populate the timestamp field,
- serializes and encrypts (if required),
- and finally increments `txseq`.

These rules apply uniformly across handshake and data packets.

2.3 Key Materials and Transcript Hashes

The implementation uses a hierarchical key structure:

- a master derivation key (`mdk`) used to derive server keys,
- a server derivation key (`sdk`) used to derive device keys,
- a device derivation key (`ddk`) provisioned to the client.

All derivations use cSHAKE with domain separation via the configuration string and the identity bytes. The server reconstructs any device key on demand using its own `sdk` and the device identity extracted from the connect request.

SKDP uses two transcript hash values:

- **dsh**, the hash of the client's connect request body,

- **ssh**, the hash of the server's connect response body.

Both values are computed using the Keccak SHA3 hash function and are used as customization strings in subsequent cSHAKE and KMAC operations. These transcript hashes bind the initial handshake messages into the entire remainder of the derivation chain.

2.4 Cipher State Initialization

During the exchange stage the client generates a random device token key `dtk`, encrypts it with a one time pad derived from `ddk` and `dsh`, and authenticates the ciphertext plus header using KMAC keyed from the same derivation stream. The server reverses this process, verifies the MAC, and recovers `dtk`. Both endpoints then derive identical client-to-server channel keys from `dtk` and `dsh`.
During the exchange response, the server generates a fresh server token `stk`, derives server-to-client channel keys from `stk` and `ssh`, encrypts `stk` with a second one time pad derived from `ddk` and `ssh`, and authenticates using KMAC. The client verifies this MAC, recovers `stk`, and initializes the server-to-client cipher state.
The resulting cipher states (`txcpr` and `rxcpr`) are used for all encryption and decryption in the establish and data phases. Every encrypted packet binds its header as associated data, providing protection for sequence numbers and timestamps.

2.5 Error Handling and Session Termination

The implementation defines explicit error codes for:

- timestamp expiration,
- MAC validation failure,
- ciphertext authentication failure,
- sequence mismatch,
- unexpected packet type.

On error, the client or server may send a one byte error code inside an error packet. The session is terminated by sending a packet with the termination flag set, closing the socket, and zeroizing all state including keys and cipher state.

2.6 Protocol Overview

The protocol consists of three phases, each containing two concrete messages. In all phases except connect, timestamps and sequence numbers are included in the associated data of the authentication layer so that replaying or modifying headers causes authentication failure.
All application traffic increments `txseq` before transmission. Handshake packets use the current `txseq` without incrementing it. On reception, `rxseq` is incremented and the incoming sequence is accepted only if it matches `rxseq` exactly.

2.7 Long Term Key Hierarchy

Before any SKDP sessions can occur, the implementation generates the long term cryptographic material that anchors all subsequent key derivation. This hierarchy consists of a master key, a server key derived from it, and device specific keys derived from the server key.

Master Key Generation The function `skdp_generate_master_key` produces a master key

$$K_{\mathsf{master}} \in \{0,1\}^{256}$$

by sampling uniformly from a cryptographically secure random number generator. This value serves as the root secret for the entire key hierarchy.

Server Key Generation Given the master key, the function `skdp_generate_server_key` derives a server specific key

$$K_{\mathsf{server}} = \text{cSHAKE}(K_{\mathsf{master}}, \mathsf{KID}_S, \text{"SKDP SERVER KEY"})$$

using cSHAKE with the server configuration string and a fixed customization label. This ensures domain separation across deployments and prevents cross protocol interference.

Device Key Generation Each device is provisioned with a device specific key

$$K_{\mathsf{device}} = \text{cSHAKE}(K_{\mathsf{server}}, \mathsf{KID}_C, \text{"SKDP DEVICE KEY"})$$

generated by `skdp_generate_device_key`. The device identity string provides namespace separation so that compromising one device key does not expose others.
These values form the long term state of the protocol. All session level keys, pads, and MAC keys are derived from these root secrets and the transcript dependent hash values described later in this section.

Connect Phase The client computes:

$$\mathsf{dsh} = H(\mathtt{CR}),$$

and the server computes:

$$\mathsf{dsh} = H(\mathtt{CR}), \qquad \mathsf{ssh} = H(\mathtt{CS}).$$

These values are stored in state and used as inputs to later KMAC and cSHAKE derivations. They are not required to match between client and server.

Connect Response The server replies with

$$\mathtt{CS} = \mathtt{KID_s} \parallel \mathtt{CFG_s} \parallel \mathtt{STOK_s},$$

using the same field layout and sizes as the client request.

Exchange Request and Response These packets contain an encrypted token bundle authenticated with KMAC. The encryption is the exclusive OR of the token with a cSHAKE derived key-stream. The serialized header is added to the KMAC state as associated data. The general form is

$$\mathtt{ciphertext} = \mathtt{token} \oplus \mathtt{keystream},$$

$$\mathtt{tag} = \text{KMAC}(\text{MAC key}, \mathtt{ciphertext} \parallel \mathtt{Header}).$$

The plaintext token is either a device token key (client to server) or a server token key (server to client). No generic AEAD construction is used here; the authenticated encryption is implemented directly through KMAC and XOR as in the reference code.

Establish Request After both sides derive the first channel key, the client generates a fresh random verification token of length `SKDP_STH_SIZE`. This value is encrypted under the client's transmit RCS state with the serialized header bound as AEAD associated data.

Establish Response The server decrypts the verification token using its receive RCS state, computes its SHA3 hash, and returns an authenticated encryption of the hash under its transmit RCS state. Successful verification confirms that both sides have derived matching channel keys.

Exchange Phase The client uses its long term device derivation key and the session hash `dsh` to derive both an encryption key and a MAC key for protecting the device token key. The server recomputes the same keys using its copy of the device derivation key and the received session hash. Correct KMAC verification and decryption confirms that both sides share the correct long term secrets and the same session hash.
Analogous operations are performed by the server when generating its encrypted server token key.

Establish Phase Using the token keys exchanged in the previous phase, both sides derive the final channel keys for the duplex RCS cipher. The client sends a random verification token encrypted under its channel 1 transmit state. The server replies with the hash of this value encrypted under channel 2 transmit. Successful verification provides explicit key confirmation.
Completion places both endpoints in the established state with symmetric channel keys for transmit and receive.

2.7.1 Client Exchange Request

This routine handles the client side of the exchange phase. It first computes the server session hash `ssh` from the server connect response, where:

$$\mathsf{ssh} = H(\mathtt{CS}) = H(\mathtt{KID_s} \parallel \mathtt{CFG_s} \parallel \mathtt{STOK_s}),$$

then generates a fresh device token key `dtk`. It encrypts `dtk` by xoring it with a cSHAKE derived key-stream and authenticates the resulting ciphertext using KMAC, with the serialized packet header bound as associated data. Finally it derives the client channel 1 transmit cipher key and nonce from `dtk` and the device session hash `dsh`.

2.7.2 Server Exchange Response

This routine is the server side of the exchange phase. It first derives the client device key `ddk` from the server key `sdk`, the configuration string and the client device identifier. It then uses `ddk` and the device session hash `dsh` to authenticate and decrypt the encrypted device token key `dtk` sent by the client. From `dtk` it derives the server channel 1 receive key and nonce. It then generates a fresh server token `stk`, derives the server channel 2 transmit key and nonce from `stk` and the server session hash `ssh`, and finally encrypts and authenticates `stk` into the exchange response using a second cSHAKE and KMAC sequence keyed by `ddk` and `ssh`.

2.7.3 Client Establish Request

This routine validates and decrypts the server exchange response, initializes the client channel 2 receive cipher, then constructs the client establish request. It encrypts a fresh random verification token, stored in the `ctx->dsh` buffer, under the client transmit cipher with the serialized header bound as associated data.

2.7.4 Server Establish Response

This routine authenticates and decrypts the client establish request using the server channel 2 receive cipher, hashes the recovered verification token, and returns its hash encrypted under the server channel 2 transmit cipher. On success it raises the session state to `skdp_flag_session_established`.

2.7.5 Client Establish Verify

This routine verifies the server establish response on the client side. It serializes the received packet header and binds it as associated data for the receive cipher, then authenticates and decrypts the ciphertext into a candidate verification hash. It recomputes the expected hash of the locally stored random verification token `dsh` and compares the two values in constant time. On success it sets the exchange flag to `skdp_flag_session_established`, otherwise it clears the flag and returns an appropriate error code.

2.7.6 Packet Encryption Procedure

The client and server packet encryption functions have identical structure. The following pseudo-code models both `skdp_client_encrypt_packet` and `skdp_server_encrypt_packet`. It checks that the session has been established, increments the transmit sequence number, assembles the header, binds it as associated data, and encrypts and authenticates the message.

2.7.7 Packet Decryption Procedure

Similarly, the client and server decryption functions have identical structure. The pseudo-code below models
`skdp_client_decrypt_packet` and `skdp_server_decrypt_packet`. It increments the receive sequence, checks that it matches the packet sequence, validates the packet time, binds the header as associated data, then authenticates and decrypts the ciphertext. On any error it returns the appropriate error code and sets the output length to zero.

2.8 Session State Machine

Both client and server maintain a state variable `exflag` that governs packet acceptance and the transitions between phases. Conceptually the protocol progresses through the stages

$$\texttt{Init} \rightarrow \texttt{Connected} \rightarrow \texttt{Exchanged} \rightarrow \texttt{Established},$$

where these stages correspond to specific `exflag` values and the presence of initialized cipher state.

- A transition to the next stage requires a valid packet with matching sequence number, a packet timestamp within the allowed window when time checks apply, and successful authentication by the relevant hash or cipher operation.
- On any failure in timestamp validation, sequence matching, authentication or message parsing, the implementation clears `exflag` and returns an error code, which aborts the handshake rather than advancing the state.
- When the establish phase completes and the verification hashes match, both client and server set `exflag` to `skdp_flag_session_established` and retain the derived channel keys for subsequent encrypted traffic.

This logical state machine reflects the behaviour of the reference code and serves as the operational basis for the protocol model used in later sections.

3 Cryptographic Model and Assumptions

This section defines the model under which SKDP is analyzed. The definitions include the system entities, the adversarial capabilities, the treatment of time, and the cryptographic assumptions required for the protocol to satisfy the security goals formalized in later sections. The model captures exactly the operational requirements visible in the engineering specification.

3.1 System Entities and Roles

The system consists of two principals, a client C and a server S, that share a long term symmetric secret used to authenticate the initial handshake. In the reference implementation this secret arises from the key hierarchy described in Section 2 (master key, server key, device key). The formal model abstracts this hierarchy as a single long term key K_{CS} shared between C and S, but correctness requires noting that the actual session specific values are derived through cSHAKE applications parameterized by the device identity, server identity, and configuration string, exactly as implemented.
Each principal maintains internal state. This includes monotonic sequence counters, local configuration parameters, device or server identity strings, and all cryptographic keys derived during the handshake. The implementation also stores two transcript hashes dsh and ssh, computed as SHA3 hashes of the raw connect request and connect response bodies, respectively. Both principals execute the same deterministic state machine described in the engineering specification, with states Init, $\mathsf{Connected}$, $\mathsf{Exchanged}$, $\mathsf{Established}$, and Error, and transitions determined strictly by packet type, timestamp validity, sequence number checks, and cryptographic verification.
A successful run of SKDP produces two authenticated channel keys: a client to server channel key K_C and a server to client channel key K_S. These keys are derived deterministically in the implementation through cSHAKE from the token keys DTK and STK combined with the transcript hashes, and they must be treated as independent keys in the security model because the derivation processes use disjoint customization strings and domain separation.

3.2 Communication and Adversarial Model

Communication takes place over an insecure network controlled by an adversary $\mathcal{A}$. The adversary may read, modify, delay, replay, drop, reorder, or fabricate packets. The adversary may start concurrent sessions, interleave messages from different executions, and perform arbitrary chosen message queries to any honest principal.
The adversary is considered active and adaptive. It may engage any principal in arbitrary sequences of protocol interactions subject only to the restrictions imposed by the state machine. The adversary learns public information such as packet headers, timestamps, and message lengths. For encrypted messages, it cannot modify any header field without causing an AEAD authentication failure, because the serialized header is always included as associated data in the underlying cipher. This reflects the exact behavior of `skdp_cipher_set_associated` in the code.
The adversary does not control the local clocks of honest principals but may attempt to influence clock readings indirectly through network based time manipulation attacks. The protocol assumes a loose time synchronization bound and uses both timestamps and sequence numbers to limit the effect of such attacks. Specifically, honest parties accept handshake messages only if their timestamp lies within a fixed window around the local clock value, matching the validity window used in the implementation via `skdp_packet_time_valid`. During normal encrypted traffic, both timestamp checks and authenticated sequence number checks apply.

3.3 Random Number Generation

The implementation uses a cryptographic random number generator to sample several values:

- the client session token STOK_C in the connect request,
- the server session token STOK_S in the connect response,
- the client device token key DTK in the exchange request,
- the server token key STK in the exchange response, and
- the client verification token STH in the establish request.

In the formal model we treat all of these as uniformly random bit strings of the appropriate length, generated by an idealized random number source. This captures the usage of `qsc_acp_generate` in the implementation. No other protocol field is required to be secret or unpredictable. All deterministic values, including identity strings, configuration strings, transcript hashes, and serialized headers, match exactly what is sent or derived in the code.

3.4 Security Assumptions

The security analysis assumes the following properties of the underlying primitives.

- The AEAD scheme satisfies IND-CPA confidentiality and INT-CTXT integrity under chosen message and chosen ciphertext attack when used with header based associated data as in the implementation. In SKDP this AEAD is instantiated either with RCS or AES GCM depending on compile time configuration, and both bind the serialized header.
- The hash function H (SHA3) is collision resistant and preimage resistant. For the purposes of transcript binding, it is also modeled as behaving like a random oracle on its inputs. This reflects its use in computing dsh and ssh.
- The KDF behaves as a pseudo random function on its key space. Outputs are computationally indistinguishable from random for an adversary without access to the long term secret or the random token material. This models the cSHAKE based derivations of DTK, STK, and the channel keys.
- Nonce values generated by honest principals, including STOK_C, STOK_S, DTK, STK, and STH, are uniformly random and remain secret until used in the protocol. These values must not repeat across sessions, matching the implementation's rejection of replayed packets through authenticated sequence numbers and timestamps.
- The time synchronization assumption noted above holds for the duration of all handshake interactions, so that honest parties reject packets whose timestamps fall outside the configured validity window. This assumption is essential because the implementation does not derive replay protection solely from nonces; timestamp based filtering appears explicitly in all handshake message processing.

These assumptions match the requirements visible in the engineering specification and the reference implementation. They are reflected explicitly in the security definitions and proofs that follow.

4 Related Work

Symmetric key distribution has been studied for several decades and underlies many deployed authentication systems. This section reviews the most relevant lines of work and positions SKDP relative to existing designs.

4.1 Classic Symmetric Key Distribution Protocols

Early work on symmetric key distribution focused on the design of protocols that allow two parties with pre-established secrets to agree on fresh session keys over an insecure network. The Needham–Schroeder symmetric protocol introduced the idea of a trusted server that distributes tickets containing freshly generated keys encrypted under long term secrets. Subsequent analysis revealed replay vulnerabilities caused by the absence of freshness guarantees on some messages, and later variants addressed these issues by including timestamps or additional nonces.

Kerberos extends this line of work into a practical authentication infrastructure. It employs a Key Distribution Center (KDC) and a Ticket Granting Service (TGS) to provide scalable distribution of time bounded tickets. Kerberos relies heavily on synchronized clocks and restricts ticket lifetimes through validity intervals encoded in the ticket contents. The architecture is three party by design: clients cannot complete a handshake without interaction with the KDC.

ISO 9798 specifies a family of entity authentication mechanisms based on symmetric and asymmetric primitives. The symmetric variants use nonces and message authentication codes to provide mutual authentication without relying on a central distribution server. These constructions influenced many later protocol designs by standardizing the use of nonces, identifiers, and message ordering as primary security tools.

SKDP departs from the Kerberos and Needham–Schroeder architecture by operating purely in a two party setting. There is no trusted third party in the protocol flow. Freshness is provided by a combination of random tokens, authenticated sequence numbers, transcript bound key derivations, and timestamp acceptance windows. All key derivation is hash based and uses cSHAKE with explicit customization strings, matching the reference implementation. This places SKDP closer to modern symmetric authenticated key exchange protocols while preserving the operational simplicity of ticket style systems.

4.2 Telecommunications Authentication and Key Agreement

Mobile telecommunications standards use symmetric key based Authentication and Key Agreement (AKA) procedures to authenticate devices and derive session keys. In these systems, a long term key is stored in a hardware token such as a SIM or USIM, and the network and the device run a protocol that produces traffic keys for subsequent encryption and integrity protection.

The AKA family uses carefully structured nonces, sequence numbers, and message authentication codes to provide mutual authentication and replay resistance. The protocols are designed to operate under strict performance and bandwidth constraints, which has led to a strong emphasis on simple key derivation chains and minimal message flows.

Architecturally, SKDP is closer to AKA than to ticket based systems. Both rely on a shared long term symmetric secret and derive traffic keys through keyed hash based functions. SKDP differs in that it does not assume hardware security modules or SIM cards and is intended to be hardware agnostic. It also differs from AKA in that SKDP derives two independent channel keys using transcript bound derivations, whereas AKA derives a small fixed set of integrity and encryption keys. SKDP separates the engineering description from the formal analysis in a way that allows the protocol to be instantiated with different symmetric primitives, including post quantum safe constructions.

4.3 Pre-Shared Key Modes in Modern Protocols

Modern transport security protocols include pre shared key (PSK) modes that allow two parties with an existing symmetric secret to establish fresh keys with low latency. TLS 1.3, for example, supports PSK based handshakes that either reuse a previously established secret or rely on out of band provisioned keys. These modes reduce the number of asymmetric operations and offer faster connection setup, which is attractive in resource constrained environments.

Datagram oriented protocols such as DTLS extend the TLS model to unreliable transports and include similar PSK options. In all of these protocols, the PSK is used in a key schedule that feeds a transcript hash into a key derivation tree. The result is a set of traffic keys that protect application data on multiple channels.

SKDP shares several ideas with these PSK modes, in particular the use of hash bound transcripts to derive multiple channel keys from a single handshake. However, SKDP is a standalone symmetric key distribution protocol rather than an extension of a general purpose transport security framework. Its message formats, fixed size headers, and explicit associated data binding reflect a design for environments where only symmetric primitives are available. SKDP also differs in that it does not use the Diffie–Hellman based hybrid modes found in TLS 1.3, and its replay protection relies on authenticated sequence numbers and timestamp windows rather than on accumulated transcript digests alone.

4.4 Positioning of SKDP

SKDP can be viewed as a two party, symmetric only analogue of the mechanisms found in Kerberos, AKA, and PSK based handshakes. It assumes a pre existing shared secret between client and server, uses a compact three stage handshake (connect, exchange, establish), and derives distinct client and server channel keys from a transcript that binds both parties' contributions. Unlike Kerberos, it does not depend on a centralized ticket server. Unlike AKA, it does not assume hardware backed key storage. Unlike TLS 1.3 PSK modes, it is not tied to a particular transport protocol or record layer.

The protocol is therefore best understood as a hash based symmetric authenticated key exchange that is suitable for systems where symmetric trust anchors are already in place and where post quantum readiness and low overhead are significant concerns. The formal analysis in the remainder of this paper is intended to place SKDP on similar theoretical footing to these established protocols while reflecting its distinct design constraints and goals.

5 Formal Protocol Specification

This section provides an abstract, protocol level specification of SKDP that is suitable for formal reasoning. The model captures exactly the values exchanged by the client and server and the deterministic derivations performed during the handshake. All notation is standard for authenticated key exchange protocols.

5.1 Notation

The following notation is used throughout this section.

- C and S denote the client and server.
- K_{CS} denotes the long term symmetric key shared by C and S. In the implementation this is realized through the master, server, and device key hierarchy, but the model abstracts it as a single logical secret.
- KID_C and KID_S denote fixed length identity strings carried in the connect messages.

- CFG_C and CFG_S denote configuration parameters (the SKDP configuration string) supplied by client and server.
- STOK_C and STOK_S denote the client and server session tokens sampled during the connect phase.
- STH denotes a fresh random verification token generated by the client during the establish phase.
- CR and CS denote the connect request and response bodies.
- $H(\cdot)$ denotes a collision resistant hash function.
- $\mathsf{AEAD.Enc}_K(AD, M)$ denotes authenticated encryption with key K, associated data AD, and plaintext M. The output is a ciphertext plus authentication tag, exactly matching the on wire format of the encrypted SKDP messages.
- $\mathsf{AEAD.Dec}_K(AD, C)$ denotes authenticated decryption which returns either the recovered plaintext or the failure symbol $\perp$.
- $\|$ denotes concatenation of byte strings.
- $\langle\text{Header}\rangle$ denotes the serialized SKDP header for the packet, containing the flag, length, sequence number, and timestamp.

All functions are deterministic given their inputs and are identical for both parties.

5.2 Long Term Key Generation

SKDP relies on a three tier hierarchy of long term keys. The formal model treats these as deterministic or random values derived from secure primitives exactly as in the implementation.

Master Key A master key

$$K_{\mathsf{master}} \overset{\$}{\leftarrow} \{0,1\}^{256}$$

is sampled uniformly from a secure random number generator. This key is assumed uncompromised for all security definitions.

Server Key The server derives a deployment specific server key

$$K_{\mathsf{server}} = \text{cSHAKE}(K_{\mathsf{master}}, \mathsf{KID}_S, \text{"SKDP SERVER KEY"}).$$

Device Key Each client device is provisioned with a device specific key

$$K_{\mathsf{device}} = \text{cSHAKE}(K_{\mathsf{server}}, \mathsf{KID}_C, \text{"SKDP DEVICE KEY"}).$$

These keys form the long term secrets from which all session level derivations originate. The formal KDF abstraction used later in the analysis corresponds directly to these cSHAKE based derivations.

5.3 Message Flow

The concrete SKDP handshake consists of six messages, organized into three phases (connect, exchange, and establish). Each message carries a fixed format header containing a type flag, body length, sequence number, and timestamp. The header is always sent in the clear and is reused as associated data in the exchange and establish phases, either via KMAC or via AEAD.

For the exchange phase it is convenient to introduce a shorthand for the MAC authenticated pad construction used in the implementation. Given keys K^{pad} and K^{mac}, associated data AD, and plaintext M, define

$$\mathsf{PadMAC}_{K^{\mathsf{pad}},K^{\mathsf{mac}}}(AD, M) = (C, T)$$

where

$$C = M \oplus \mathsf{Pad}(K^{\mathsf{pad}}), \qquad T = \mathsf{KMAC}(K^{\mathsf{mac}}, C \parallel AD).$$

In SKDP the pad and MAC keys are derived from the long term derivation keys and the transcript hashes using cSHAKE, exactly as in the reference implementation.

Concrete Engineering Flow. The reference implementation executes the following message sequence:

$$\begin{aligned}
&C \to S : \mathsf{CR} = \mathsf{KID}_C \parallel \mathsf{CFG}_C \parallel \mathsf{STOK}_C && \text{(connect request)}\\
&S \to C : \mathsf{CS} = \mathsf{KID}_S \parallel \mathsf{CFG}_S \parallel \mathsf{STOK}_S && \text{(connect response)}\\
&C \to S : \mathsf{XR} = (C_{\mathsf{XR}}, T_{\mathsf{XR}}) = \mathsf{PadMAC}_{K^{\mathsf{pad}}_{\mathsf{XR}},K^{\mathsf{mac}}_{\mathsf{XR}}}(\langle \mathsf{Header} \rangle, \mathsf{DTK}) && \text{(exchange request)}\\
&S \to C : \mathsf{XS} = (C_{\mathsf{XS}}, T_{\mathsf{XS}}) = \mathsf{PadMAC}_{K^{\mathsf{pad}}_{\mathsf{XS}},K^{\mathsf{mac}}_{\mathsf{XS}}}(\langle \mathsf{Header} \rangle, \mathsf{STK}) && \text{(exchange response)}\\
&C \to S : \mathsf{ER} = \mathrm{AEAD.Enc}_{K_C}(\langle \mathsf{Header} \rangle, \mathsf{STH}) && \text{(establish request)}\\
&S \to C : \mathsf{ES} = \mathrm{AEAD.Enc}_{K_S}(\langle \mathsf{Header} \rangle, H(\mathsf{STH})) && \text{(establish response).}
\end{aligned}$$

Here:

- STOK_C and STOK_S are fresh client and server session tokens,
- DTK and STK are the device and server token keys,
- STH is the client verification token in the establish phase,
- $K^{\mathsf{pad}}_{\mathsf{XR}}, K^{\mathsf{mac}}_{\mathsf{XR}}$ and $K^{\mathsf{pad}}_{\mathsf{XS}}, K^{\mathsf{mac}}_{\mathsf{XS}}$ are cSHAKE derived pad and MAC keys for the exchange messages, computed from the long term derivation keys and the transcript hashes $\mathsf{dsh}, \mathsf{ssh}$,
- K_C and K_S are the AEAD channel keys derived from DTK, STK, and the transcript hashes as specified by the key schedule,
- $\langle \mathsf{Header} \rangle$ is the serialized SKDP header.

The exchange messages XR and XS are not AEAD ciphertexts. They are protected by the PadMAC construction, in which the header is bound as associated data via KMAC. The establish messages ER and ES are AEAD encrypted, with the serialized header bound as associated data in the cipher.

Abstract Cryptographic Flow. For the purposes of formal reasoning, the six engineering messages are grouped into conceptual transcript stages:

$$\begin{aligned}
&C \to S : \mathsf{CR}\\
&S \to C : \mathsf{CS}\\
&C \to S : (C_{\mathsf{XR}}, T_{\mathsf{XR}}) = \mathsf{PadMAC}_{K^{\mathsf{pad}}_{\mathsf{XR}},K^{\mathsf{mac}}_{\mathsf{XR}}}(\langle \mathsf{Header} \rangle, \mathsf{DTK})\\
&S \to C : (C_{\mathsf{XS}}, T_{\mathsf{XS}}) = \mathsf{PadMAC}_{K^{\mathsf{pad}}_{\mathsf{XS}},K^{\mathsf{mac}}_{\mathsf{XS}}}(\langle \mathsf{Header} \rangle, \mathsf{STK})\\
&C \to S : \mathrm{AEAD.Enc}_{K_C}(\langle \mathsf{Header} \rangle, \mathsf{STH})\\
&S \to C : \mathrm{AEAD.Enc}_{K_S}(\langle \mathsf{Header} \rangle, H(\mathsf{STH})).
\end{aligned}$$

These six messages constitute the full transcript used in the AKE partnering definition and the security proofs.

Key Confirmation and Termination. The client accepts the session only if the final establish response verifies correctly. Writing ES for the received establish ciphertext, this means:

$$\mathrm{AEAD.Dec}_{K_S}(\langle\mathsf{Header}\rangle, \mathsf{ES}) = H(\mathsf{STH}),$$

where both K_S and $H(\mathsf{STH})$ depend on the connect and exchange transcripts through the cSHAKE based key schedule.
Both parties transition to the *Established* state only after processing their final message successfully. Any failure at any step (timestamp validation, sequence number check for encrypted messages, KMAC verification in the exchange phase, AEAD authentication in the establish phase, or hash comparison on the client) terminates the session and enters the error state, matching the reference implementation.

6 Security Definitions

This section defines the security properties that SKDP aims to achieve. The definitions follow standard models for authenticated key exchange and include confidentiality, authentication, key confirmation, replay resistance, and limited privacy guarantees. The model reflects the transcript structure, key derivation steps, and state transitions specified in the formal protocol description and implemented in the reference code.
Let a protocol instance be an ordered pair (P, i) where $P \in \{C, S\}$ and $i \in \mathbb{N}$ indexes the session. Each instance maintains its local state, sequence numbers, transcript hashes $\mathsf{dsh}, \mathsf{ssh}$, and derived keys exactly as specified earlier. The long term keys used by SKDP are assumed to be generated securely using the functions described in Section 2 and remain uncompromised for the duration of the security experiment.

6.1 Authenticated Key Exchange Security

Authenticated key exchange security captures two requirements. First, that an honest session derives its keys only when communicating with the intended peer. Second, that any derived keys agree with the keys derived by the peer in the matching session.
Two sessions (C, i) and (S, j) are *partners* if their transcripts match in order and content, meaning:
1. They processed connect messages whose bodies produce the same transcript hashes $\mathsf{dsh} = H(\mathsf{CR})$ and $\mathsf{ssh} = H(\mathsf{CS})$.
2. They each verified the MAC-protected exchange messages $(C_{\mathsf{XR}}, T_{\mathsf{XR}})$ and $(C_{\mathsf{XS}}, T_{\mathsf{XS}})$ under the PadMAC construction derived from their respective long term keys and transcript hashes.
3. They each accepted the AEAD-protected establish messages, including the final encrypted value $H(\mathsf{STH})$.
These conditions ensure that both endpoints used the same long term keying material, transcript hashes, and exchanged token keys, exactly as in the implementation.

Definition 1 (Authenticated Key Exchange)**.** A protocol satisfies authenticated key exchange security if, with all but negligible probability over the randomness of honest principals:

- Whenever (C, i) or (S, j) outputs a session key pair (K_C, K_S), it has a unique partner session.
- No two distinct sessions output the same channel keys unless they are partners. In particular, uniqueness holds because the key derivation uses the transcript dependent hashes dsh and ssh.

- If the adversary does not interfere with the handshake, both parties output identical key pairs derived from the same token material.

This definition excludes adversarially induced key mismatches and ensures that honest principals agree on the session key material exactly as produced by the SKDP key schedule.

6.2 Key Indistinguishability

Key indistinguishability requires that the session keys produced by an honest session are indistinguishable from random to any adversary that does not break the long term secret or the underlying primitives. For SKDP this means the adversary must not forge, learn, or modify the token keys DTK or STK, nor forge the PadMAC tags on the exchange messages, nor decrypt or modify the AEAD-protected establish messages.
The definition is expressed using a standard left or right indistinguishability experiment.

Definition 2 (Key Indistinguishability)**.** Let (P, i) be an honest completed session with partner (P', j). Define the game KI as follows:

- The challenger samples a bit b.
- If $b = 0$, it returns the real key pair (K_C, K_S) derived by the session.
- If $b = 1$, it returns a pair of uniformly random strings of the same lengths.
- The adversary must guess b.

A protocol achieves key indistinguishability if the adversary's advantage in this game is negligible in the security parameter.

This definition captures confidentiality of the derived keys under adaptive adversarial interaction and reflects the fact that, in the implementation, the keys are deterministically derived from random token material and transcript bound hashes.

6.3 Key Confirmation

SKDP includes an explicit key confirmation step. The client sends a fresh verification token STH encrypted under its channel key K_C, and the server responds with an encrypted hash of that token under K_S. Successful decryption ensures correctness of the derived keys on both sides.

Definition 3 (Implicit Key Confirmation)**.** A protocol provides implicit key confirmation if an honest party that derives a session key can be assured that its peer could have derived the same key, assuming the peer is not corrupted.

Implicit key confirmation follows from the deterministic nature of the SKDP key derivation functions and transcript binding. Both sides compute their keys from the same token material and session hashes once the exchange phase is complete.

Definition 4 (Explicit Key Confirmation)**.** A protocol provides explicit key confirmation if, whenever an honest party accepts the final message, it is assured that its peer actually derived the same session key and successfully computed the confirmation value.

In SKDP, the final encrypted message containing $H(\mathsf{STH})$ provides explicit key confirmation. The client accepts the session only if the decrypted value equals the locally computed hash $H(\mathsf{STH})$, matching the implementation.

6.4 Replay and Reordering Resistance

Replay of exchange messages is prevented by the KMAC authentication in the PadMAC construction. A replayed $(C_{\mathsf{XR}}, T_{\mathsf{XR}})$ or $(C_{\mathsf{XS}}, T_{\mathsf{XS}})$ will be rejected because the timestamp fails the connect/exchange acceptance window or because the MAC does not validate under the current transcript hash. Reordering of exchange messages is prevented by the strict state machine ordering.

Definition 5 (Replay Resistance)**.** A protocol provides replay resistance if no adversary can cause an honest party to accept a message that was previously accepted in any session, unless the message is part of the same transcript and uses the expected sequence number.

In SKDP, replay of encrypted packets is prevented by requiring

$$\texttt{pkt.sequence} = \texttt{rxseq},$$

and by rejecting any message whose timestamp falls outside the acceptance window when applicable. The implementation increments `rxseq` only after successful authentication, ensuring strict replay protection.

Definition 6 (Reordering Resistance)**.** A protocol provides reordering resistance if an adversary cannot cause an honest party to accept messages out of order in a way that leads to incorrect session state transitions.

All encrypted SKDP packets bind their sequence number and timestamp into the AEAD associated data, so any modification results in authentication failure. Handshake packets rely solely on timestamp validation and their fixed order in the state machine.
The timestamp condition:

$$|T_A - T_B| \leq 60 \text{ seconds}$$

ensures protection from replay of stale handshake packets, matching the constant `SKDP_PACKET_TIME_THRESHOLD` in the implementation.

6.5 Privacy and Identity Protection

SKDP does not provide full identity hiding, because the connect messages contain unencrypted identity fields KID_C and KID_S. Some limited privacy is achieved ... because later transcript elements and all token bundles are protected under PadMAC (cSHAKE pad plus KMAC tag) and do not leak information beyond ciphertext length.

Definition 7 (Partial Identity Protection)**.** A protocol provides partial identity protection if an adversary cannot learn any party's long term secret or derived session keys from observing its identity fields or other publicly visible information.

SKDP satisfies this property under the assumptions on the hash function, the AEAD primitive, and the cSHAKE based key derivation functions, even though the identities themselves are transmitted in the clear.

7 Provable Security Analysis

This section provides a proof oriented analysis of the security properties defined earlier. The arguments follow a standard reduction based strategy in which any adversary that breaks a protocol property is transformed into an adversary that breaks one of the underlying cryptographic primitives. The proofs reflect the deterministic transcript binding, the hash based token derivations, and the authenticated encryption structure of SKDP. The analysis follows the actual implementation, which uses MAC protected one time pad encryption for the exchange phase and AEAD only for the establish and data phases.

7.1 Overview of the Reduction Strategy

The security of SKDP follows from four observations:

1. Both the client and server contribute fresh randomness to each session, including STOK_C, STOK_S, DTK, STK, and STH.
2. All cryptographically sensitive values in the exchange phase are protected by a PadMAC construction, consisting of a cSHAKE-derived one-time pad and a KMAC authentication tag with the serialized header bound as associated data. Any modification of C_{XR} or C_{XS} or their tags is detected with overwhelming probability.
3. The establish phase uses AEAD with the serialized header included as associated data. Confidentiality and integrity of token bundles through cSHAKE derived pads and KMAC authentication, and confidentiality of the confirmation messages through AEAD.
4. The transcript hashes dsh and ssh bind the connect messages into all subsequent cSHAKE expansions that derive pad keys, MAC keys, and channel keys. Any tampering changes these hashes and causes later MAC or AEAD checks to fail.

These observations yield a reduction to the PRF security of cSHAKE, the unforgeability of KMAC, and the confidentiality and integrity of the AEAD primitive.

7.2 Confidentiality of Session Keys

This section provides the proof sketch for key indistinguishability. The goal is to show that an adversary cannot distinguish the real channel keys from randomly chosen keys, except with negligible probability.

Theorem 1 (Key Indistinguishability)**.** *If the PadMAC construction used in the exchange phase is unforgeable, the AEAD scheme used in the establish phase is IND-CPA and INT-CTXT secure, the cSHAKE-based KDF behaves as a PRF on its key input, and H is collision resistant, then the session keys K_C and K_S produced by SKDP are indistinguishable from random for any adversary who does not know the long term secret.*

Proof Sketch. Let $\mathcal{A}$ distinguish real channel keys from random with non-negligible advantage. We define the following hybrids.

Hybrid 0. The real SKDP execution matching the implementation exactly: PadMAC-protected exchange, AEAD-protected establish, cSHAKE-derived pads, MAC keys, and channel keys.

Hybrid 1. Replace the PadMAC outputs $(C_{\mathsf{XR}}, T_{\mathsf{XR}})$ and $(C_{\mathsf{XS}}, T_{\mathsf{XS}})$ with values generated by an ideal PadMAC oracle. Any difference in $\mathcal{A}$'s view breaks the unforgeability of the KMAC tag or the PRF security of the cSHAKE-derived pad. This corresponds exactly to forging or distinguishing the exchange messages in the real protocol.

Hybrid 2. Replace the AEAD encryptions in the establish phase with outputs from an IND-CPA oracle. Because the serialized header is included as associated data and the channel keys are unknown to the adversary, any distinguishing advantage reduces to AEAD IND-CPA security.

Hybrid 3. Replace all cSHAKE-derived keys:

$$K_{\mathsf{XR}}^{\mathsf{pad}},\, K_{\mathsf{XR}}^{\mathsf{mac}},\, K_{\mathsf{XS}}^{\mathsf{pad}},\, K_{\mathsf{XS}}^{\mathsf{mac}},\, K_C,\, K_S$$

with outputs of random functions keyed on unknown secret material. Any successful distinguishing attack contradicts the PRF security of cSHAKE.

Hybrid 4. Replace dsh and ssh with uniformly random strings unless a collision occurs. Such collisions occur with negligible probability under the collision resistance of H.
In Hybrid 4, the distribution of (K_C, K_S) is identical to the distribution of uniformly random strings. No efficient adversary can distinguish Hybrid 4 from Hybrid 0 except with negligible probability. □

Theorem 2 (Explicit Key Confirmation). *Because* DTK *and* STK *are established by MAC-protected PadMAC messages, and because the establish phase uses AEAD with the header bound as associated data, the server can recover* STH *only if it derived the same cSHAKE-based channel key* K_C*. Likewise the client accepts only if the AEAD decryption under* K_S *yields the correct hash* $H(\mathsf{STH})$*.*

Proof Sketch. The client samples a random verification token STH and encrypts it under its transmit channel key K_C using the AEAD scheme with the serialized header as associated data. The server can recover STH only if it derived the same key schedule inputs and therefore the same value K_C.
After computing $H(\mathsf{STH})$, the server encrypts this value under its transmit channel key K_S, again binding the header as associated data.
The client accepts only if the decrypted value equals $H(\mathsf{STH})$. Any incorrect value results in AEAD rejection or hash mismatch. Producing a valid ciphertext under the wrong key contradicts AEAD ciphertext integrity. Producing a matching hash without knowing STH contradicts collision resistance of H.
Thus the final message proves that both sides derived identical channel keys and completed the same transcript. □

7.3 Replay Protection and Timestamp Usage

Replay protection for the exchange phase relies on timestamp filtering for the connect-stage transcripts and KMAC authentication for the PadMAC exchange messages. Replay protection for the establish and data phases relies on both AEAD integrity and strict sequence number enforcement, as the serialized header is authenticated as associated data.

Theorem 3 (Replay and Reordering Protection). *Assuming the time synchronization bound, integrity of the KMAC authenticated exchange messages, and AEAD ciphertext integrity for the establish and application messages, an adversary cannot cause an honest participant to accept a replayed or reordered message in any SKDP session.*

Proof Sketch. Replay protection in SKDP operates in two layers.

Handshake messages. Connect and exchange messages are not AEAD protected. Instead, the receiver verifies that the timestamp lies within the allowed window. Any replayed handshake message fails timestamp validation unless it arrives within this narrow window, and any modification to message contents is detected by recomputing and verifying the transcript dependent MAC (exchange phase) or rejecting on identity mismatch (connect phase).

AEAD protected messages. For establish and application messages, the receiver increments rxseq before checking the incoming sequence number. If the packet does not match the expected value, it is rejected without attempting decryption. Because AEAD binds the entire header, including the sequence number and timestamp, an adversary cannot modify these fields or produce a new valid ciphertext.

Replaying a previously accepted application ciphertext fails either the timestamp check (when applicable) or the sequence number check. Reordering valid ciphertexts changes their sequence order and causes immediate rejection. These properties follow exactly from the decrypt routines in the implementation. □

7.4 Composition with Generic AEAD

The protocol is designed so that its security depends only on standard AEAD and hash based assumptions, and not on any internal structure of the specific cipher used. The exchange phase uses a MAC authenticated XOR pad, while the establish phase relies on AEAD.

Theorem 4 (AEAD Agnostic Composition)**.** *If the AEAD scheme provides IND-CPA confidentiality and ciphertext integrity, and the MAC used in the exchange phase is unforgeable, then SKDP achieves authenticated key exchange security for any instantiation of these primitives that satisfies these properties.*

Proof Sketch. All sensitive information in the establish phase is protected by AEAD with the serialized header as associated data. This includes the verification token STH and $H(\mathsf{STH})$. Thus:

- confidentiality of STH reduces to AEAD IND-CPA security,
- integrity of all AEAD protected messages reduces to ciphertext integrity,
- confidentiality and authenticity of DTK and STK reduce to the pseudo-randomness of the cSHAKE derived pads and unforgeability of the KMAC tag in the exchange phase,
- transcript binding arises from the deterministic hashing that feeds dsh and ssh,
- replay and reordering resistance follow from authenticated sequence numbers in AEAD messages and timestamp checks in handshake messages.

Because the reduction depends only on these abstract security properties, any standard AEAD primitive and unforgeable MAC can be used in place of the reference implementation without affecting the correctness of the formal analysis. □

8 Cryptanalytic Evaluation

This section provides a cryptanalytic evaluation of SKDP based on the protocol structure, the underlying primitives, and the adversarial model. The evaluation focuses on the attack surfaces exposed by the design and discusses how the protocol defends against each class of attack. The goal is not to introduce new formal proofs, but to highlight practical considerations that complement the reduction based analysis.

8.1 Attack Surfaces and Design Choices

The primary attack surfaces of SKDP fall into four categories: token handling, transcript binding, misuse of AEAD encryption, and manipulation of timestamp based checks.

Token Handling The device and server tokens STOK_C and STOK_S are central to the intermediate key derivation process. Exposure or modification of these values would alter all derived keys. SKDP prevents such attacks by protecting the token bundles DTK and STK using a cSHAKE derived one time pad followed by a KMAC authentication tag. Any attempt to tamper with a token bundle results in a KMAC verification failure during the exchange phase, forcing the recipient into the error state. Token confidentiality relies on the pseudo-randomness of the cSHAKE derived pad and the secrecy of the long term device derivation key.

Long Term Key Generation The functions `skdp_generate_master_key`, `skdp_generate_server_key`, and `skdp_generate_device_key` implement the key hierarchy used in the formal model. The master key is generated using a secure RNG, while the server and device keys are derived using cSHAKE with fixed customization strings. The formal abstraction of these keys as K_{CS} corresponds exactly to the tuple $(K_{\mathsf{master}}, K_{\mathsf{server}}, K_{\mathsf{device}})$ as realized in the reference code.

Transcript Binding The transcript hash values $\mathsf{dsh} = H(\mathsf{CR})$ and $\mathsf{ssh} = H(\mathsf{CS})$ bind the identities, configuration strings, and initial tokens into deterministic inputs to the KDF. Modifying or replaying a connect message yields a different transcript hash and therefore different cSHAKE output blocks for pad generation and key derivation. As a result, any forged message produces incorrect MACs or mismatched AEAD keys in later phases, causing authentication failure.

AEAD Misuse Only the establish and data messages in SKDP are encrypted under AEAD, not the exchange messages. The protocol uses associated data correctly by binding the serialized header into all AEAD invocations. Because the header contains the message type flag, length, sequence number, and timestamp, the AEAD tag authenticates all of these values.
This prevents message substitution, truncation, or reordering of ciphertext blocks in the establish or application phases. Since no nonce is reused with the same key—the nonce is derived deterministically per message from the cipher state—standard misuse resistant requirements for AEAD usage are satisfied.

Timestamp Manipulation Timestamps in handshake messages are not authenticated by AEAD, because handshake packets are unencrypted. Instead, the implementation applies timestamp validation as an external freshness check. An adversary cannot force acceptance of a stale handshake packet unless it falls within the configured acceptance window. For encrypted packets, the timestamp is authenticated as part of the AEAD associated data, preventing tampering. Replayable messages are rejected either by timestamp window violation or by sequence number mismatch.

8.2 Analysis of Timestamps vs Counters

Replay protection can be implemented through timestamps or monotonic counters. SKDP uses both: timestamps provide early filtering in handshake messages, and authenticated sequence numbers enforce strict message ordering for all AEAD protected packets.

Timestamp Advantages Timestamps provide protection against immediate replay of handshake messages and allow devices to discard stale attempts without performing any cryptographic work. The implementation uses a sixty second window reflecting realistic clock drift bounds. For AEAD protected messages, timestamps are authenticated and cannot be modified.

Timestamp Limitations Relying on clocks introduces risks. A device with an inaccurate clock may reject valid handshake messages or accept replayed ones that appear to fall within the window. An adversary controlling a time synchronization channel may attempt to shift a device's clock. However, timestamps serve only as auxiliary checks. The primary replay protection mechanism is the sequence number in AEAD protected packets and the MAC verification for exchange messages.

Sequence Number Enforcement Every AEAD protected packet must match the expected sequence value and must authenticate correctly under the derived key. The decrypt routines increment `rxseq` before checking the incoming sequence number; any mismatch causes immediate rejection without decryption. This rule eliminates replay and reordering attacks even under clock drift or timestamp manipulation. The exchange phase inherits replay protection from the KMAC authenticated transcript.

8.3 Key Compromise and Forward Security Considerations

Because SKDP is a symmetric only protocol derived from a long term secret, it cannot provide full forward secrecy. Compromise of the long term shared key K_{CS} allows an adversary to recompute the server and device derivation keys and therefore derive all token pads, MAC keys, and channel keys for past sessions.

Effect of Token Exposure Exposure of the ciphertext token bundles DTK or STK alone does not reveal session keys, because they are generated from random material XOR padded with pseudo-random blocks. However, if an adversary obtains ddk or the long term keying inputs from which it is derived, it can recompute the entire pad sequence and validate MACs, fully recovering the session.

Protection of Past Sessions While SKDP cannot achieve forward secrecy against long term secret compromise, it does protect past session content under AEAD confidentiality. An adversary must first break AEAD confidentiality to learn the verification token or application payloads. The transcript hashes and derived keys remain secure as long as the long term secret and cSHAKE derived values are uncompromised.

Potential Extensions Forward secrecy could be added by incorporating a public key ephemeral value or a symmetric one time secret shared out of band. Such extensions fall outside the core design goals of SKDP but could be considered for future versions where forward secrecy is required.

8.4 Denial of Service and State Exhaustion

The SKDP state machine follows a rigid transition order and rejects invalid messages at the earliest possible stage. Nevertheless, several resource focused attacks remain possible.

Malformed Packets Packets with invalid lengths, timestamps outside the acceptance window, or incorrect sequence numbers are rejected before AEAD processing. This protects against computational exhaustion attacks that rely on forcing repeated attempts to decrypt malformed ciphertext.

AEAD Verification Cost Although AEAD decryption occurs only after header validation, an attacker can still impose load by sending many well formed but unauthenticated ciphertexts. Implementations may employ rate limiting or simple network filtering to mitigate these attacks.

State Desynchronization Because sequence counters are incremented only after successful authentication, an attacker cannot desynchronize the endpoints by injecting invalid messages. Attempts to advance counters incorrectly instead result in immediate errors without side effects.

Session Reset and Abort Any authentication or parsing failure causes an immediate transition to the error state, preventing the protocol from entering unintended or partially initialized states. Resource usage is bounded by the number of simultaneous incomplete sessions an implementation chooses to track.
This analysis indicates that SKDP is robust against standard classes of practical attacks and that its security reduces cleanly to the strength of the underlying cryptographic primitives and the enforcement of timestamp and sequence based protections.

9 Implementation Alignment

This section evaluates the correspondence between the formal protocol specification, the SKDP protocol specification document, and the reference implementation. The goal is to demonstrate that the engineering level behavior of the protocol matches the abstract model used for the security analysis. Where differences exist, we clarify how the formal model interprets the intended semantics of the implementation.

9.1 Comparison with the SKDP Specification

The SKDP specification document defines the protocol at a descriptive level, including message formats, session token usage, and the general structure of the connect, exchange, and establish phases. The formal model of SKDP presented in this paper matches the structure of the specification with the following observations.

Message Structure The specification describes a three phase handshake with identifiers, configuration parameters, and session token fields in the connect messages. The formal model uses the same fields, abstracted as fixed length byte strings. The specification lists token handling and transcript hashing as core mechanisms, both of which appear directly in the formal protocol description.

Session Hashes The specification defines the device and server session hashes as $H(\mathsf{CR})$ and $H(\mathsf{CS})$. The formal model retains this exact definition. The session hashes serve as transcript binders and are used as inputs to the KDF in the same manner as described in the specification.

Key Hierarchy The specification states that session keys are derived from the long term secret, configuration parameters, and session tokens using hash based derivations. The formal model uses a generic KDF abstraction that captures this behavior while remaining independent of implementation specific details. This aligns with the actual implementation, which uses cSHAKE expansions keyed by the derivation keys and transcript hashes.

Replay Protection The specification describes replay protection based on timestamps and sequence numbers. The formal model incorporates both mechanisms and treats the timestamp bound as an explicit assumption. As in the implementation, handshake packets rely on timestamp filtering, while encrypted packets rely on authenticated sequence numbers.
Overall, the formal model matches the SKDP specification with no structural differences.

9.2 Comparison with the Reference Implementation

The reference implementation defines the precise behavior of SKDP through the functions in `skdp.c`, `skdpclient.c`, and `skdpserver.c`. The formal protocol matches this implementation by reflecting the logic of packet handling, key derivation, and state transitions. The main points of alignment are summarized below.

Header Layout The implementation serializes each packet header as

$$\texttt{flag} \parallel \texttt{msglen} \parallel \texttt{sequence} \parallel \texttt{utctime}$$

in little endian format. The same layout appears in the engineering specification and is assumed in the formal model. This ensures that the packet header used as AEAD associated data is identical across the model and implementation for all AEAD protected messages.

Session Hash Definitions The implementation computes

$$\mathsf{dsh} = H(\mathsf{CR}) \quad \text{and} \quad \mathsf{ssh} = H(\mathsf{CS}),$$

where CR and CS are the raw message bodies. The formal specification uses the same definitions. There are no additional fields or ordering changes in the implementation that would affect transcript binding.

Token and Nonce Handling Token bundles are handled using `skdp_cipher_transform`, which implements a cSHAKE derived one time pad combined with a KMAC authentication tag. This is not an AEAD construction, so the formal model represents it using a MAC authenticated pad abstraction rather than an AEAD interface. The correction aligns the model with the implementation.

The session nonce STH is generated randomly by the client and is protected using AEAD under the client channel key. The server returns $H(\mathsf{STH})$ encrypted under the server channel key, completing key confirmation exactly as described in the formal model.

Error Handling Paths The implementation immediately aborts a session if any of the following occur:

- a timestamp is outside the allowed window,
- a sequence number does not match the expected value (for AEAD protected packets),
- the MAC tag fails verification in the exchange phase,
- the AEAD authentication tag fails in the establish phase,
- a message has an invalid length or structure.

The formal state machine includes an Error state that captures this behavior abstractly. There are no code paths that contradict the formal semantics.

State Machine Alignment The client and server implementations progress through the states Init, $\mathsf{Connected}$, $\mathsf{Exchanged}$, and $\mathsf{Established}$. These states match the abstract model exactly. In both descriptions, transitions occur only after successful authentication of each message, and no computational shortcuts or implicit acceptance conditions exist.

Based on this evaluation, the formal model is a faithful abstraction of the reference implementation.

9.3 Constant Time and Side Channel Considerations

The reference implementation uses hash based constructions and deterministic encryption routines that do not involve branching on secret values. The code paths associated with token processing, transcript hashing, pad generation, and KDF evaluation are constant time for inputs of fixed length. This limits, but does not eliminate, potential timing channels.

Timing Variations Packet header parsing and timestamp checks may introduce observable timing variation because failure may occur before cryptographic processing. In the exchange phase, MAC verification only occurs after header checks and sequence validation. These timing differences do not leak session keys or token material, but may reveal whether a packet was rejected before or after cryptographic authentication. Implementations that require stronger timing resistance may reorder checks or add uniform delay.

Memory Safety and Zeroization The implementation clears sensitive buffers such as token bundles, derived keys, pad blocks, and intermediate state using explicit zeroization routines. While this reduces exposure risk, memory safety depends on the correctness of the deployment environment. Side channels arising from memory reuse or hardware effects are outside the scope of the protocol model.

Randomness Considerations The values requiring high quality randomness are the session nonce STH and the session tokens STOK_C and STOK_S. Weak randomness may affect the strength of the confirmation step or allow partial prediction of session tokens. The long term secret and derivation keys are not randomized at runtime, so a robust entropy source is recommended for the random values generated during the handshake.
These considerations indicate that the protocol is suitable for constant time and memory safe implementations, and that potential side channel risks are limited and well understood given the symmetric design.

10 Performance and Practicality

This section evaluates the practical aspects of SKDP, including its message complexity, computational cost, and performance relative to well established symmetric key distribution mechanisms. The goal is to highlight the suitability of SKDP for constrained systems, embedded deployments, and environments that rely exclusively on symmetric primitives.

10.1 Asymptotic Cost

The asymptotic cost of SKDP is dominated by hash evaluations, small fixed length KDF operations, and authenticated encryption of the confirmation messages. Token bundles are protected by a cSHAKE derived one time pad followed by a KMAC authentication tag, not by AEAD, so their cost resembles that of a short keyed hash operation and XOR masking. All operations rely solely on symmetric primitives with running times linear in the input size.

Message Complexity SKDP uses a six message handshake at the engineering level (connect request and response, exchange request and response, establish request and response). Abstracting the exchange round into the key derivation subroutine, this can be viewed as four logical protocol flows $C \to S, S \to C, C \to S, S \to C$, that is, one round trip plus a final confirmation message.

Computational Complexity Each party performs:

- one hash of the connect message,
- one derivation of an intermediate key from the long term secret,
- one pad generation and KMAC verification for the received token bundle,
- one pad generation and KMAC computation for the transmitted token bundle,
- one hash of the session nonce and one AEAD encryption or decryption of the confirmation value.

The only AEAD operations occur during the establish phase and later during application data transfer. Since the confirmation payloads are small, the AEAD cost is minimal. The dominant cost of the exchange phase comes from cSHAKE and KMAC, both lightweight hashing primitives. The total computation is constant per handshake and independent of application data size.

Round Trip Latency The protocol requires one full round trip for handshake completion. This is similar to TLS 1.3 in PSK mode and comparable to many mobile network AKA handshakes. In practice the latency is governed by network conditions rather than computational cost.

10.2 Implementation Cost

The implementation cost of SKDP is low because it uses only symmetric primitives with fixed length inputs and minimal state. This simplifies deployment in resource constrained systems.

Computational Requirements Token encryption, transcript hashing, KMAC authentication, and KDF evaluations require a small number of Keccak permutation calls. On typical embedded hardware these operations complete in a few hundred microseconds. On general purpose hardware they are negligible.

Memory Footprint The reference implementation maintains:

- a cipher context used for AEAD operations in the establish and data phases,
- temporary buffers for token bundles, transcript hashes, and header serialization,
- a compact state structure containing sequence counters, timestamps, configuration values, and identities.

All buffers are fixed size. No dynamic memory allocation is required during the handshake. This is advantageous in embedded environments because it eliminates fragmentation risk and simplifies memory management.

Code Footprint The protocol implementation consists of a few thousand lines of C code, including token handling, cSHAKE and KMAC routines, AEAD support, header serialization, and state machine logic. The implementation does not depend on external libraries beyond basic cryptographic primitives and can be integrated into systems with minimal runtime environments.

10.3 Comparison with Existing Schemes

SKDP can be compared with several existing symmetric key distribution mechanisms.

Kerberos Kerberos relies on a trusted ticket server and uses timestamps for replay protection. Session keys are delivered through encrypted tickets rather than derived directly from peer contributions. SKDP differs in that it operates without a third party and binds all handshake messages into a shared transcript. This makes SKDP more suitable for systems where a centralized authority is unavailable. In terms of performance, SKDP avoids the overhead associated with ticket generation and reduces message size by removing the need for nested encryption.

TLS PSK TLS 1.3 in PSK mode uses a one round trip handshake similar to SKDP, with a transcript hash that feeds the key schedule. Both rely exclusively on symmetric primitives when operating in PSK only mode. SKDP is simpler, because it does not include complex extensions, cipher suite negotiation, or a record layer. For systems that do not need a general purpose transport protocol, SKDP offers lower code complexity and reduced attack surface.

Mobile Network AKA Protocols The AKA family uses symmetric KDF chains and explicit authentication codes. Like SKDP, AKA relies on a long term secret and produces traffic keys for secure communication. SKDP differs in that it does not assume secure hardware storage or telecommunications specific infrastructure. Its message structure is more general and does not require operator specific fields. The computational cost is similar, but SKDP provides a more generalized, application level handshake suitable for embedded devices outside the telecommunications environment.
Overall, SKDP achieves performance that is competitive with well established symmetric handshake mechanisms while maintaining a much smaller code footprint and a simpler design. This makes it attractive for systems that require efficient symmetric only authenticated key exchange with clear formal security guarantees.

11 Limitations and Future Work

This section outlines the limitations of the present analysis and identifies areas where SKDP or its modeling framework may be refined. Some limitations arise from the protocol design itself, while others reflect choices made in the formal model.

11.1 Modeling Limitations

The formal model abstracts the behavior of the underlying cryptographic primitives and treats them as idealized components with well defined security properties. Several practical considerations are therefore outside the scope of the analysis.

Side Channel Leakage The model does not account for side channel attacks such as timing variation, power analysis, or microarchitectural leakage. Although the implementation uses constant time operations for fixed length inputs and avoids branching on secret material during token processing and KDF evaluation, practical deployments may involve hardware or system level effects that are not captured in the formal treatment. A more detailed evaluation of physical leakage would require a separate analysis framework.

Time Synchronization The model assumes loose clock synchronization between honest parties. It does not address the mechanisms by which clocks are synchronized or the security of such mechanisms. Attacks that influence local time through network interference are partially mitigated by sequence numbers for AEAD protected messages and by KMAC verification in the exchange phase, but a full treatment of secure time distribution remains outside the scope of this work.

Resource Constraints The model assumes that principals have sufficient computational and memory resources to maintain session state and perform token processing. Extremely constrained devices may require additional optimizations that are not reflected in the present analysis.

11.2 Protocol Limitations

The design of SKDP reflects the constraints of a symmetric only environment. While the protocol achieves strong confidentiality and authentication under these constraints, it does not provide all properties that may be desirable in broader applications.

Lack of Forward Secrecy SKDP does not provide forward secrecy against long term key compromise. An adversary who obtains the shared secret K_{CS} can recompute the derivation keys ddk and sdk, regenerate the pads used to encrypt token bundles, and recreate all channel keys for recorded sessions. This limitation is inherent in symmetric only protocols and is common among deployed mechanisms such as Kerberos and mobile network AKA procedures.

Identity Exposure The connect messages transmit identities and configuration parameters in the clear. Although later messages are encrypted or MAC protected, the protocol does not provide identity hiding against passive adversaries during the initial handshake. In environments where identity privacy is required, additional measures may be necessary.

Lack of Key Update Mechanisms SKDP produces a single pair of channel keys per handshake. The protocol does not include a mechanism for periodic key updates or for deriving new traffic keys from an existing session. Applications requiring long lived secure channels may need to provide or compose a separate key update mechanism.

11.3 Directions for Improvement

Several extensions and refinements may address the limitations noted above and serve as potential directions for future work.

Ephemeral Key Contributions Forward secrecy could be introduced by adding an ephemeral contribution to the transcript, for example through a post quantum key encapsulation mechanism or a symmetric one time secret distributed out of band. Such enhancements would require extending the formal model to capture the resulting transcript binding and key schedule changes.

Adaptive Time Handling Time synchronization may be improved by combining timestamps with signed time beacons or monotonic counters. These approaches can reduce reliance on loosely synchronized clocks and may strengthen replay resistance in environments with unstable time sources.

Key Update Extensions A lightweight key update procedure could be added to SKDP by deriving additional traffic keys from the existing channel state using a hash based key ratchet. Such mechanisms can provide post compromise recovery and limit the exposure of long lived keys.

Identity Protection Mechanisms Identity confidentiality could be improved through encryption of the connect messages using a pre established secrecy key or through the use of pseudonyms. These extensions would require additional shared secrets or a more complex key management framework.

The design of SKDP strikes a balance between simplicity, performance, and formal verifiability. The potential extensions outlined above provide avenues for enhancing its security properties while preserving its symmetric only structure.

12 Conclusion

12.1 Summary of Results

This paper presented the design and formal analysis of the Symmetric Key Distribution Protocol (SKDP), a lightweight authenticated key exchange mechanism intended for systems that rely exclusively on symmetric cryptography. The protocol was specified in an engineering accurate manner based on the reference implementation, including precise definitions of packet structure, transcript binding, and deterministic key derivation. A formal model was introduced that captures authenticated key exchange security under a powerful network adversary, and reduction based proofs were given for confidentiality, authentication, replay protection, and explicit key confirmation.

A comparison with the SKDP specification and the implementation demonstrated that the formal model faithfully represents the intended behavior of the protocol. The analysis also positioned SKDP within the landscape of symmetric key distribution mechanisms and argued that its design offers practical advantages for constrained and post quantum ready environments.

12.2 Security Guarantees and Caveats

Under standard assumptions for hash based key derivation, MAC unforgeability for the exchange phase, and secure authenticated encryption for the establish and application phases, SKDP provides the following guarantees:

- confidentiality and integrity of token bundles through cSHAKE derived pads and KMAC authentication, and AEAD protection of the confirmation messages,
- uniqueness of the derived channel keys for each completed handshake through the transcript bound key schedule,
- mutual authentication through transcript binding and explicit key confirmation,
- replay and reordering resistance based on sequence numbers for AEAD protected messages and timestamp filtering for handshake messages.

These guarantees hold in the presence of an active adversary that controls the network but does not compromise the long term shared secret.

Several caveats apply. SKDP does not provide forward secrecy because the handshake relies solely on a symmetric long term key and deterministic derivation chains. Compromise of this key enables regeneration of all past and future session keys. The protocol does not hide identities in the connect messages, and its replay protection for handshake messages depends on loosely synchronized clocks. These limitations are inherent in the symmetric only design and are shared by many practical systems in this category.

12.3 Outlook

SKDP offers a simple and efficient foundation for symmetric authenticated key exchange that can be incorporated into a range of secure communication environments. Its lightweight construction makes it suitable for embedded systems, distributed control networks, and other scenarios where symmetric trust anchors are already present. The protocol can be extended with additional mechanisms such as key update procedures or ephemeral contributions if stronger security properties are required.

Future work may explore compositions of SKDP with higher level authenticated encryption schemes, secure messaging layers, or symmetric post quantum constructs. Because the protocol is fully based on hash functions, MAC based authenticated encryption for the exchange phase, and AEAD for the establish phase, it aligns well with long term migration strategies that emphasize symmetric security. SKDP therefore represents a practical and formally grounded approach to authenticated key exchange in environments where symmetric cryptography is the preferred foundation.

References

1. Needham, R. M., and Schroeder, M. D. *The Use of Encryption for Authentication in Large Networks of Computers.* Communications of the ACM, 1978. Available at: https://doi.org/10.1145/359657.359659
2. Otway, D., and Rees, O. *Efficient and Timely Mutual Authentication.* ACM Operating Systems Review, 1987. Available at: https://doi.org/10.1145/37499.37500
3. Kohl, J., and Neuman, C. *The Kerberos Network Authentication Service (V5).* IETF RFC 4120, 2005. Available at: https://www.rfc-editor.org/rfc/rfc4120
4. 3GPP. *3G Security, Security Architecture.* 3GPP TS 33.102, 2023. Available at: https://www.3gpp.org/ftp/Specs/archive/33_series/33.102/
5. 3GPP. *Security Architecture and Procedures for 5G System.* 3GPP TS 33.501, 2023. Available at: https://www.3gpp.org/ftp/Specs/archive/33_series/33.501/
6. Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3.* IETF RFC 8446, 2018. Available at: https://www.rfc-editor.org/rfc/rfc8446
7. IETF. *Datagram Transport Layer Security (DTLS) Version 1.3.* IETF RFC 9147, 2022. Available at: https://www.rfc-editor.org/rfc/rfc9147
8. NIST. *FIPS 202: SHA-3 Standard, Permutation-Based Hash and Extendable-Output Functions.* U. S. Department of Commerce, 2015. Available at: https://doi.org/10.6028/NIST.FIPS.202
9. NIST. *SP 800-185: SHA-3 Derived Functions, cSHAKE, KMAC, TupleHash, and ParallelHash.* U. S. Department of Commerce, 2016. Available at: https://doi.org/10.6028/NIST.SP.800-185
10. Bellare, M., and Rogaway, P. *Entity Authentication and Key Distribution.* Advances in Cryptology, CRYPTO 1993. Available at: https://cseweb.ucsd.edu/~mihir/papers/eakd.pdf
11. Bellare, M., and Rogaway, P. *The Authenticated Key Exchange Security Model.* UCSD Lecture Notes, widely cited reference. Available at: https://cseweb.ucsd.edu/~mihir/papers/kemain.pdf
12. Krawczyk, H. *SIGMA: The SIGn-and-MAc Approach to Authenticated Diffie Hellman.* CRYPTO 2003. Available at: https://webee.technion.ac.il/~hugo/sigma.pdf

13. Underhill, J. G. *SKDP Protocol Specification.* Quantum Resistant Cryptographic Solutions Corporation, 2025. Available at: https://www.qrcscorp.ca/documents/skdp_specification.pdf.

14. Underhill, J. G. *SKDP Reference Implementation (C Source Code).* Quantum Resistant Cryptographic Solutions Corporation, 2025. Available at: https://github.com/QRCS-CORP/SKDP.