Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 28 August 2026

A. Asadi
Individual

24 February 2026

TCP In-Band Single Packet Authentication (TCP-SPA)
draft-asadi-tcp-spa-01

Abstract

This document describes a mechanism called TCP In-Band Single Packet Authentication (TCP-SPA). A client that wishes to open a TCP connection to a protected server embeds a compact Message Authentication Code (MAC) inside the TCP SYN packet itself, carried as an experimental TCP option (kind 253, per [RFC6994]), with an ExID (Experiment Identifier) value of TBD to be assigned by IANA. The server verifies the MAC at the earliest possible point in the network stack — before a socket is allocated or the TCP handshake proceeds — and drops the SYN if verification fails.

This approach differs from existing Single Packet Authentication (SPA) systems in that no out-of-band authentication packet is required: authentication and connection establishment are a single atomic step.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 August 2026.

Copyright Notice

Table of Contents

1. Introduction and Problem Statement

Internet-facing TCP services are continuously probed by scanners, bots, and opportunistic attackers. Even when a service is properly hardened — strong passwords, TLS, privilege separation — the act of completing or even initiating a TCP handshake consumes server resources: socket allocation, kernel scheduler wakeups, TLS state machine initialization. At scale, or under a targeted SYN-flood, this cost becomes significant.

The ideal property would be: a server never allocates any resource for a connection until it has cryptographic proof that the remote party holds a valid secret. In particular, the proof should arrive in the very first packet — the TCP SYN — so that an illegitimate SYN can be discarded at the lowest possible layer without ever touching the socket layer.

Additional desiderata:

a. The mechanism MUST NOT require any out-of-band communication channel, pre-flight packet, or extra round trip. Authentication and connection establishment must be a single atomic operation from the client’s perspective.

b. The mechanism MUST be transparent to the application. A standard connect(2) call, with no modification to the application, should automatically carry the authentication token.

c. The mechanism MUST be NAT traversal-friendly. Many legitimate clients are behind one or more Network Address Translators. Any mechanism that binds authentication to the client’s IP address creates a timing race when NAT rewrites the address and breaks address-bound tokens.

d. The mechanism SHOULD allow key rotation without service interruption.

e. The cryptographic overhead per SYN MUST be small enough to run inside a high-speed packet processing path (e.g., before socket allocation) without measurable latency impact.

No existing IETF-standardized mechanism satisfies all five properties simultaneously. Section 2 explains why.

2. Why Existing Mechanisms Do Not Solve This Problem

2.1. TCP MD5 Signature Option (RFC 2385)

[RFC2385] defines a TCP option (kind 19) that carries an MD5 signature over the full TCP segment, the IP pseudo-header, and a shared secret. It was designed to protect BGP sessions between routers in a known topology.

It fails the stated requirements for several reasons:

NAT incompatibility (violates property c): The MAC input includes the source and destination IP addresses from the IP pseudo-header. A NAT device that rewrites the source address after the client has

computed the signature will cause verification to fail at the server. There is no way to make [RFC2385] work through NAT without defeating its security guarantee.

Wrong threat model (different problem): [RFC2385] is designed to prevent session hijacking on a long-lived BGP session between two known peers. It authenticates every segment, including data segments, and requires both peers to share the same key for the lifetime of the session. The problem addressed here is authenticating the initiating SYN from an arbitrary client before any session state is created. These are fundamentally different problems.

MD5 is no longer considered secure: MD5 is cryptographically broken. [RFC5925] was written specifically to replace [RFC2385].

2.2. TCP Authentication Option, TCP-AO (RFC 5925)

[RFC5925] replaces [RFC2385] with a more modern construction. It supports multiple MAC algorithms, master keys and traffic keys (via a key derivation function defined in [RFC5926]), and key rotation.

It still fails the stated requirements:

NAT incompatibility (violates property c): Like its predecessor, TCP-AO includes the IP source and destination addresses in the MAC computation (Section 5.1.3 of [RFC5925], the "connection identifier"). A NAT that rewrites the source address breaks authentication. [RFC5925] explicitly notes (Section 7) that NAT is incompatible with TCP-AO unless the NAT also updates the MAC, which is impossible without the shared key — defeating the purpose.

Designed for peer authentication, not port authorization (different problem): TCP-AO secures an established TCP session between two long-lived peers (typically routers or servers) whose addresses and keys are known in advance. The problem addressed here is pre-authentication of arbitrary clients attempting to initiate a new connection, where the client population changes dynamically and may be behind NAT.

Requires kernel support and configuration on both ends: TCP-AO must be configured via socket options before the connection is established. This requires application modification (violates property b) or OS-level transparent interception, which TCP-AO was not designed to support.

In summary: TCP-AO is an excellent solution to the problem it was designed for (securing BGP and similar sessions between known peers). It is not a solution to pre-connection port authorization for arbitrary, NAT-traversing clients.

Protection during connection vs. pre-connection: TCP-AO provides authentication for all segments during an established connection, protecting against session hijacking and data injection. TCP-SPA, by contrast, authenticates only the initial SYN packet before any connection state is created. This enables resource-efficient filtering of illegitimate connection attempts at the earliest possible point, before socket allocation or TCP state machine initialization. The two mechanisms address different threat models: TCP-AO protects established connections, while TCP-SPA prevents resource consumption from connection attempts that should never be allowed to establish.

Port hiding: Because the server silently drops any SYN that does not carry a valid authentication token, a protected port produces no observable response to unauthorised probes. To a port scanner or automated reconnaissance tool, the port appears closed or filtered. TCP-AO and [RFC6978] provide no such property: a TCP-AO-protected port still completes (or at least begins) the handshake before authentication fails, making the service discoverable. Port hiding is an inherent consequence of TCP-SPA's drop-on-SYN design, not an add-on feature.

2.3. TCP-AO NAT Extension (RFC 6978)

[RFC6978] extends TCP-AO ([RFC5925]) to operate through Network Address Translators. It does so by defining a modified key derivation function that excludes the IP addresses from the MAC computation, making the authentication NAT-traversal-friendly. This is the closest existing mechanism to TCP-SPA, and it is worth explaining precisely where the two differ.

[RFC6978] secures an entire TCP session — every segment, from SYN to FIN — between two configured peers. TCP-SPA authenticates only the initial SYN before any connection state is created, and imposes no overhead on subsequent segments. The goal is not session integrity but pre-connection port authorization: preventing illegitimate SYNs from ever reaching the TCP stack. RFC 6978 solves NAT compatibility for TCP-AO's use case; TCP-SPA solves a different problem.

2.4. Out-of-Band SPA (e.g., fwknop)

The most widely deployed SPA implementation is fwknop [FWKNOP]. Its protocol, while not an IETF standard, is the reference point for the SPA concept. The model is:

1. The client sends a specially crafted UDP packet (or ICMP packet, or similar) to the server. This packet contains a token: a MAC over the client's IP address, a timestamp, and a shared secret.

2. The server's SPA daemon (running in userspace, monitoring a raw socket or pcap) receives the packet, verifies the token, and temporarily installs a firewall rule to open the target port for the source IP of the packet.

3. The client then initiates a normal TCP connection.

This fails properties (a) and (c):

Requires an extra round trip (violates property a): Steps 1-2 must complete before step 3 can succeed. The client needs a separate tool (not just connect(2)) to send the SPA packet. There is an unavoidable timing window between firewall-rule installation and the TCP SYN arriving.

Source-IP binding is NAT-hostile (violates property c): The firewall rule opened in step 2 is keyed to the source IP seen in the SPA UDP packet. If the client is behind a NAT that uses a different source IP for UDP and TCP traffic (e.g., due to NAPT hairpinning or load balancing), the TCP SYN will arrive from a different address than the one whitelisted, and the connection will be refused. In practice, fwknop deployments work around this by accepting a timing window long enough to account for NAT, which widens the attack surface.

Plaintext source IP in the token (additional concern): The token explicitly encodes the client's IP address. This means the authenticating data is not independent of NAT-rewritten fields.

2.5. Port Knocking

Port knocking predates SPA. The client sends a sequence of packets to a sequence of closed ports; the server observes the sequence and opens a port.

It is entirely unauthenticated in the cryptographic sense: any observer who captures the sequence can replay it. Some variants add encryption, but the fundamental fragility remains. Port knocking is not considered a serious candidate for the requirements in Section 1 and is mentioned only for completeness.

3. Proposed Solution: TCP In-Band SPA

TCP In-Band SPA embeds the authentication token inside the TCP SYN packet as a TCP option, using kind 253 (the experimental kind permanently allocated by [RFC6994]), with an ExID value of TBD (to be assigned by IANA) to avoid collisions with other experiments.

The high-level model is:

1. A client that wants to connect to a protected server computes a SipHash-2-4 MAC over a small input struct that includes a coarse timestamp, a Key ID, and the TCP Initial Sequence Number (ISN) of the SYN.

2. The client writes this MAC, along with the Key ID, version, and timestamp, into a TCP option field of the outgoing SYN. This is done transparently at the packet processing layer, before the packet leaves the host, requiring no application modification.

3. The server, at the earliest possible ingress processing point (before socket allocation), reads the TCP option, looks up the corresponding key by Key ID, recomputes the expected MAC, and either passes or drops the SYN.

Note: This specification defines the protocol mechanism independent of any particular implementation. While the reference implementation uses eBPF (XDP for server-side, TC egress for client-side), the protocol itself can be implemented using any programmable packet processing framework or even traditional kernel modifications. The eBPF implementation serves as a proof of concept demonstrating the feasibility of early-packet authentication.

This satisfies all five properties from Section 1:

a. *Single step:* the SYN itself carries the proof. No pre-flight packet is needed.

b. *Transparent:* the option is injected at the packet processing layer before the packet leaves the host; no application change is required.

c. *NAT friendly:* the MAC input does NOT include the source IP address. A NAT that rewrites the source IP does not invalidate the MAC. The TCP ISN, which is carried in the SYN and is not rewritten by standard NAT, ties the token to the specific connection.

d. *Key rotation:* the Key ID field allows multiple keys to coexist. A client can use a new key immediately; the server keeps old keys until they expire.

e. *Lightweight:* SipHash-2-4 over 14 bytes is a handful of XOR and rotate operations, well within the cycle budget of a high-speed packet processing path.

4. TCP Option Format

The SPA option is carried as a TCP option with kind 253 (experimental, per [RFC6994]) and the following layout:

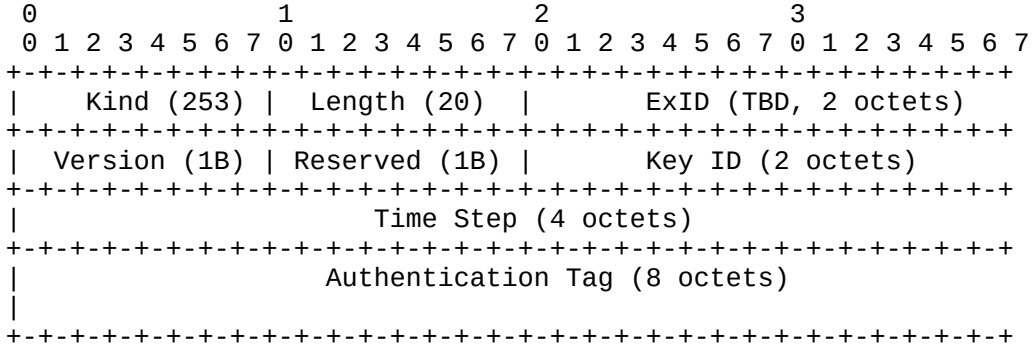

Figure 1

Total option length: 20 octets.

Fields:

Kind (1 octet) 253, the experimental TCP option kind permanently allocated by [RFC6994].

Length (1 octet) 20 (total option length including Kind and Length octets).

ExID (2 octets) Experiment Identifier, network byte order.

Disambiguates this experiment from other users of kind 253, per [RFC6994]. The value is TBD (pending IANA assignment from the ExID sub-registry defined by [RFC6994]).

Version (1 octet) Protocol version. MUST be 1 in this version of the specification.

Reserved (1 octet) MUST be zero on send; ignored on receive.

Key ID (2 octets) Selects which pre-shared key the sender used. Allows the receiver to maintain multiple keys simultaneously for rotation purposes. Network byte order.

Time Step (4 octets) A coarse, monotonically increasing counter used for replay prevention. The granularity is implementation-defined; a step of 30 seconds is RECOMMENDED. Network byte order.

Authentication Tag (8 octets) The truncated SipHash-2-4 output as described in Section 5.

5. Hash Input and MAC Computation

The Authentication Tag is computed as follows:

```
struct spa_input {
    uint16_t  exid;       /* ExID, network byte order         */
    uint16_t  ver;        /* Version                           */
    uint16_t  key_id;     /* Key ID, network byte order        */
    uint32_t  time_step;  /* Time Step, network byte order     */
    uint32_t  tcp_seq;    /* TCP ISN from the SYN, net order   */
};

tag = SipHash-2-4(k0, k1, spa_input)
```

Figure 2

where k0 and k1 are the two 64-bit halves of the 128-bit pre-shared key identified by Key ID.

The TCP ISN (tcp_seq) is included so that the tag is cryptographically bound to this specific SYN and cannot be replayed in a different connection with a different ISN.

The source IP address is deliberately excluded from the hash input. Including it would break authentication through NAT (see Section 10).

SipHash-2-4 is defined in [SIPHASH]. It produces a 64-bit output. All 64 bits are used as the Authentication Tag.

6. Client Behavior

The client MUST:

1. Select a Key ID and retrieve the corresponding 128-bit pre-shared key.
2. Obtain a current Time Step value.
3. Generate a normal TCP SYN, allowing the TCP stack to select an ISN.
4. Intercept the outgoing SYN before transmission (e.g., at the TC egress hook).
5. Populate the spa_input struct with the chosen ExID, version, Key ID, Time Step, and the TCP ISN from the SYN header.
6. Compute the Authentication Tag.
7. Prepend the tcp_opt_spa option to the existing TCP option list, shifting other options forward.
8. Update tcphdr.doff to reflect the added option bytes.
9. Recompute the TCP checksum and IP total length.

The client SHOULD check that the resulting TCP header does not exceed 60 octets (the maximum allowed by the 4-bit doff field). If the option list is already full, the client MAY omit less-critical options (e.g., TCP timestamp) to make room, or MAY fail open (send without the SPA option) if configured to allow it.

Note: The 40-octet option space limit is a fundamental constraint of the TCP header format (RFC 793). The TCP-SPA option was deliberately designed to fit within 20 octets — half the available budget — to leave room for commonly used options such as MSS, SACK Permitted, Window Scale, and Timestamps. Ongoing work in the TCPM working group (see [I-D.ietf-tcpm-tcp-edo]) proposes an Extended Data Offset (EDO) mechanism that would allow TCP headers to exceed 60 octets, relaxing this constraint for future versions of this or similar specifications. TCP-SPA does not depend on EDO and operates within the current 40-octet limit.

7. Server Behavior

The server MUST:

1. At the earliest possible processing point — before socket or connection state is allocated — inspect all incoming TCP packets.

2. For each TCP SYN destined to a protected (IP, port) pair:

    1. Search the TCP option list for an option with kind 253 and the expected ExID (TBD). If absent, DROP the SYN.

    2. Extract Key ID from the option.

    3. Look up the pre-shared key by (destination, Key ID). If not found, DROP the SYN.

    4. Populate spa_input from the option fields and the SYN's ISN.

    5. Compute the expected tag and compare it to the tag in the option. If they differ, DROP the SYN.

    6. Validate the Time Step: it MUST be within a configurable acceptance window of the server's current time (e.g., +/- 1 step). If outside the window, DROP the SYN (replay or excessively stale packet).

    7. Pass the SYN to the normal TCP stack for further processing.

3. For non-SYN packets (including ACK, data, FIN) destined to protected addresses, the server SHOULD pass them without re-checking the SPA option. The SPA option is not present in non-SYN segments.

4. For TCP SYN+ACK (the server's handshake reply), no SPA option is required.

8. Replay Prevention

The Time Step field provides coarse replay prevention. A captured SYN with a valid SPA option can be replayed during the acceptance window. To reduce this risk:

* The acceptance window SHOULD be as small as clock skew and network delay allow (one step, e.g., 30 seconds, is RECOMMENDED).

* The TCP ISN binds the tag to a specific SYN. An attacker who replays a captured SYN will be replaying an exact packet including the ISN; the server's TCP stack will likely reject the duplicate SYN through standard ISN validation, even if the SPA tag passes.

* Implementations MAY maintain a short-lived cache of recently seen (Key ID, Time Step, ISN) tuples to detect and reject exact duplicates within the acceptance window.

9. Key Management

This document does not define a key exchange protocol. Pre-shared keys MUST be distributed through an out-of-band secure channel.

The Key ID field (16 bits) allows up to 65535 distinct keys per (destination, server) pair. This is sufficient for key rotation: a new key is distributed before the old one expires, and the server accepts both Key IDs during the overlap period.

10. NAT Considerations

Standard (port-preserving) NAPT rewrites the source IP address and possibly the source TCP port. This implementation deliberately excludes both from the MAC computation, so NAPT does not break authentication.

The TCP ISN is not rewritten by standard NAT. If a middlebox were to rewrite the ISN (ISN randomization on NAT, which some NAT implementations do), the MAC would fail. This is a known limitation. Implementations SHOULD document this.

The SPA TCP option itself is not rewritten by any known NAT implementation; TCP options other than the Timestamp option ([RFC7323]) are generally passed through unmodified. However, some deep-packet-inspection (DPI) middleboxes may strip unknown TCP options. In environments where this is a concern, the SPA option MUST be placed as the last option before EOL so that stripping it does not corrupt other options.

11. Security Considerations

Authentication strength: SipHash-2-4 with a 128-bit key produces a 64-bit tag. An attacker who does not know the key has a 2^-64 probability of forging a valid tag for a given (Key ID, Time Step, ISN) tuple. At line rate this is a negligible risk; at 1 Mpps the expected forgery time exceeds 584,000 years. SipHash was designed for exactly this use case (short-input PRF) and has withstood public cryptanalysis since 2012 [SIPHASH].

Key secrecy: The security of this mechanism depends entirely on the secrecy of the pre-shared key. Keys MUST be generated with a cryptographically secure random number generator and distributed via a confidential, integrity-protected channel.

Denial-of-service: The server-side verification runs before any socket or kernel TCP state is allocated. Illegitimate SYNs are dropped at the cost of a key lookup and a SipHash computation — both are O(1) and extremely fast. This mechanism is therefore more DoS-resistant than approaches that require socket allocation before authentication.

Port hiding: A protected port produces no observable response to SYNs that do not carry a valid authentication token — they are silently dropped before any handshake begins. This makes the service invisible to port scanners and automated reconnaissance. Operators SHOULD be aware that this property relies on the server's packet filter running before the host TCP stack; if the filter is bypassed or disabled, the port becomes visible.

Option stripping: An on-path attacker who strips the SPA option from a SYN would cause the connection to be dropped by the server (which expects the option on protected destinations). This is a denial-of- service capability, not an authentication bypass. Replay: see Section 8.

Key ID enumeration: An attacker can observe which Key IDs are used (the field is not encrypted) and attempt brute-force against a specific Key ID. The 64-bit tag size makes this computationally infeasible within any realistic time window.

12. IANA Considerations

This document uses TCP option kind 253, which is permanently allocated for experimental use by [RFC6994] and requires no IANA action. [RFC6994] defines an ExID sub-registry; this document requests assignment of an ExID value (TBD) from that registry to identify the TCP-SPA experiment and avoid collisions with other users of kind 253.

No other IANA actions are required.

13. References

13.1. Normative References

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998, <https://www.rfc-editor.org/rfc/rfc2385>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010, <https://www.rfc-editor.org/rfc/rfc5925>.

[RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal", RFC 6978, August 2013, <https://www.rfc-editor.org/rfc/rfc6978>.

[RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010, <https://www.rfc-editor.org/rfc/rfc5926>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, August 2013, <https://www.rfc-editor.org/rfc/rfc6994>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", RFC 7323, September 2014, <https://www.rfc-editor.org/rfc/rfc7323>.

13.2. Informative References

[SIPHASH] Aumasson, J-P. and D. J. Bernstein, "SipHash: a fast short-input PRF", INDOCRYPT 2012, 2012, <https://www.aumasson.jp/siphash/siphash.pdf>.

[I-D.ietf-tcpm-tcp-edo] Touch, J. and W. Eddy, "TCP Extended Data Offset Option", Work in Progress, Internet-Draft, draft-ietf-tcpm-tcp-edo, 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-tcp-edo>.

[FWKNOP] Rash, M., "Single Packet Authorization with Fwknop", USENIX LISA 2006, 2006, <https://www.usenix.org/legacy/events/lisa06/tech/rash.html>.

Author’s Address

Amin Asadi
Individual
Email: aminassadi.og@gmail.com