ESP-Ping

Implementation

Antony Antony

July 16, 2026

Overview

Implementation of IETF IPsecME Drafts. draft-ietf-ipsecme-encrypted-esp-ping-03

draft-ietf-ipsecme-esp-ping : use IP-TFS wireformat instead of I.D.

Linux kernel implementation: Majority of the code is in `net/ipv4/esp_ping.c`, `net/ipv6/esp6_ping.c` `net/xfrm/xfrm_esp_ping.c`

UAPI most changes are in: `include/uapi/linux/in.h` `include/uapi/linux/xfrm.h` `include/uapi/linux/ip.h`

Why ESP Ping? A re-cap

Encrypted ESP Ping (post-IKE, inside an established SA):

- authenticated liveness check, per SA / per path
- `return_spi` targets one specific path when multiple SAs exist
  - e.g. primary/backup path — wired vs 4G/5G failover
  - e.g. RFC9611 multipath scenarios
- IKEv2 capability negotiation → silence reliably means path failure

Plain ESP Ping (SPI 7/8, pre-IKE):

- tests raw ESP (protocol 50) reachability before IKE negotiation.
- unauthenticated, no ICV — proves the protocol passes, nothing more
- no capability negotiation — silence is ambiguous

Three modes ESP Ping

Mode	Outer ESP SPI	Inner payload
Plain	7 / 8 (fixed)	`esp_echo_hdr` + data, unencrypted
Mode 1	SA SPI	encrypted AGGFRAG `esp_echo_hdr` + data
Mode 2	SA SPI	encrypted AGGFRAG sub_type=1 (CC). Deferred to IP-TFS CC implementation

- **Plain**: IANA-reserved SPI values, no SA, no decryption — kernel auto-responds
- **Mode 1**: new AGGFRAG subtypes, peer announces support in IKE_AUTH(ENCRYPTED_PING_SUPPORTED), commits to respond as best effort.[1]
- **Mode 2**: reuses existing IPTFS CC subtype, IKEv2 need USE_AGGFRAG

[1] Subject to local policy and resource availability, rate limiting, etc.

AGGFRAG sub-type dispatch

Byte 0 of payload (same field as `ip_iptfs_hdr.subtype`)

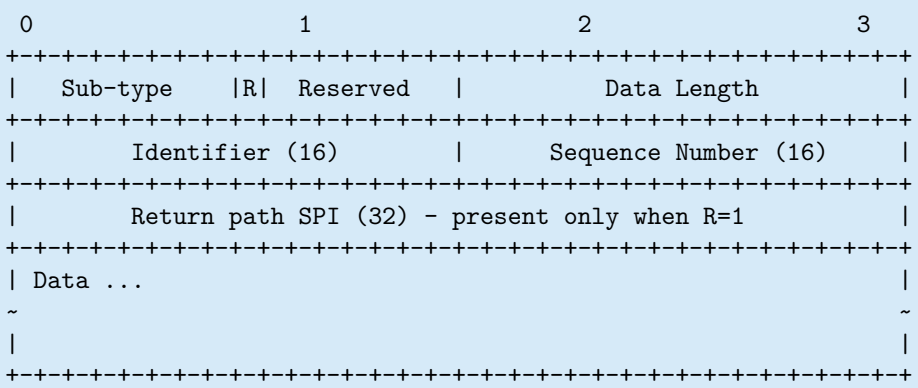
Sub-type	Format
1 (CC)	`ip_iptfs_cc_hdr` then probe data
TBD2	`esp_echo_hdr` — echo request
TBD3	`esp_echo_hdr` — echo response

Full encrypted packet:

`IP | ESP hdr | encrypted(esp_echo_hdr [| return_spi] | data ) | ESP auth tag`

ESP Next Header = `IPPROTO_AGGFRAG` (144). No inner IP header.

Wire format (identical for plain & encrypted. Drafts would merge?)



esp_echo_hdr

```
struct esp_echo_hdr {
    __u8    sub_type;   /* AGGFRAG dispatch byte */
    __u8    flags;      /* bit 7: R flag (return_spi present) */
    __be16  data_len;   /* payload bytes */
    __be16  id;         /* echo id, similar to ICMP echo id */
    __be16  seq;        /* sequence number */
};
    /* __be32 return_spi follows when R=1 */
```

Fields Detail

`return_spi` present only when R=1:

- request → which SA the responder should use for the reply
- response → echoed back so the initiator can validate it
- **not used for plain SPI 7/8** — reply path is fixed (always SPI=8 back to source)

`id`:

- encrypted ping: kernel fills from `inet_sport`
- plain ping: sender fills directly

SA flag: XFRM_SA_XFLAG_ESP_PING

Mode 2 needs the kernel SA when the peer negotiated ENCRYPTED_PING_SUPPORTED.

props.flags (u8) is full → use props.extra_flags (u32, XFRMA_SA_EXTRA_FLAGS):

```
#define XFRM_SA_XFLAG_OSEQ_MAY_WRAP     2  /* bit 1, existing */
#define XFRM_SA_XFLAG_ESP_PING          4  /* bit 2: SA accepts Encrypted ESP echo */
```

Set by IKE daemon: `ip xfrm state add ... extra-flag esp-ping` On both directions, in and out.

Enforced at 3 points: esp_ping_deliver_request(), esp_ping_deliver_response(), esp_ping_v4_sendmsg() (→ -EPERM if missing)

IKEv2 : negotiating ESP-ping

vici child option: `esp_ping = yes` (both peers)

```
Initiator (OPT_ESP_PING)                Responder (OPT_ESP_PING)
  CREATE_CHILD_SA + ENCRYPTED_PING_SUPPORTED  -->
                                          echoes ENCRYPTED_PING_SUPPORTED
                          <--  ENCRYPTED_PING_SUPPORTED
  both: child_sa->set_esp_ping() -> XFRM_SA_XFLAG_ESP_PING on in+out SA
```

- `ENCRYPTED_PING_SUPPORTED` notify = 40960 (private-use placeholder, pending IANA)
- verify: `swanctl --list-sas --raw` → `esp-ping=yes`
- test: `testing/tests/ikev2/net2net-esp-ping/`

Socket design

`AF_INET, SOCK_DGRAM, IPPROTO_ESP = 50` — same analogy as ICMP ping:

```
ESP  = 50 -> wire protocol AND socket(AF_INET, SOCK_DGRAM, IPPROTO_ESP)

ICMP = 1  -> wire protocol AND socket(AF_INET, SOCK_DGRAM, IPPROTO_ICMP)
```

- **Initiator** (ping tool): `sendto()`, kernel fills `id = inet_sport`, response demuxed by `id`
- **Responder** (strongswan): `bind()`, no `connect()` — kernel delivers requests to any open unconnected socket
- **Optional SPI override**: `setsockopt(IP_ESP_PING_SPI)` pins send path to a specific SA

Listener opt-in — request fan-out

Mirrors `raw_v4_input()`'s clone-to-all delivery (`net/ipv4/raw.c`):

```
/* listen to all esp-ping SAs */
setsockopt(fd, IPPROTO_IP, IP_ESP_PING_LISTEN, NULL, 0);

/* or filter to specific SAs only */
__be32 spis[] = { spi_a, spi_b };
setsockopt(fd, IPPROTO_IP, IP_ESP_PING_LISTEN, spis, sizeof(spis));
```

- Empty array, or array containing SPI `0` → wildcard (listen to all)
- Non-empty specific list → filtered delivery, checked before `skb_clone()`
- Not calling this setsockopt → non-listener (the default)
- Lets a real strongSwan responder **and** a passive monitor both get their own clone of every request

Why listening is a separate opt-in

Not inferred from `spi_out` or `sk_state`, because:

- An **initiator** (BusyBox `ping -E`) wants only its own matched response via `esp_ping_deliver_response()`'s per-id lookup — regardless of whether it pins an SA via `IP_ESP_PING_SPI` or relies on the SPD
- Conflating "which SA to pin for sending" with "do I want the fan-out" would silently mis-route requests to initiators that don't pin an SPI
- Sockets default to non-listening → BusyBox needs no change to stay excluded

Future extension: generalize to "watch decrypted ESP traffic for any SPI," not just echo — needs hooking `esp_input`/`xfrm_input` directly, a separate larger piece of work.

Architecture — send path (encrypted)

```
userspace sendmsg(fd, [esp_echo_hdr, data], daddr)
  esp_ping_v4_sendmsg(sk, msg, len)
    - validate esp_echo_hdr, sub_type
    - route + get SA (SPD or pinned spi_out)
        check x->props.extra_flags & XFRM_SA_XFLAG_ESP_PING
    - alloc skb, build AGGFRAG payload
        echo_h->id = inet->inet_sport   <- kernel fills id
    - IPCB(skb)->flags |= IPSKB_XFRM_AGGFRAG_PAYLOAD
    - xfrm_output(sk, skb)
        -> esp4_output: encrypts, prepends ESP (NH=AGGFRAG)
        -> xfrm4_prepare_output -> xfrm4_aggfrag_encap_add()
```

Architecture — receive path (encrypted)

```
ESP packet arrives -> esp4_input() decrypts
  xfrm_input(): inner next-header = IPPROTO_AGGFRAG (144)
    AGGFRAG dispatch: read sub_type = skb->data[0]
      0 (BASIC) -> existing IPTFS path
      1 (CC)    -> existing IPTFS CC path
      TBD2 -> esp_ping_deliver_request(net, x, skb)
      TBD3 -> esp_ping_deliver_response(net, x, skb)
```

- **deliver_request**: any open unconnected esp-ping socket (*swan)
- **deliver_response**: lookup by `id` (16-bit, keyed by `inet_sport`) → initiator

Delivered skb starts at `esp_echo_hdr` — no prefix headers.

Socket roles: the four operations

`esp_ping_v4_sendmsg()`/`recvmsg()` are symmetric (like ICMP echo) — role differs only in which delivery function feeds `recvmsg()`.

Operation	Delivery mechanism	Who
send request	n/a (outbound)	BusyBox `ping -E`
receive response	per-id lookup, one socket	BusyBox `ping -E`
receive request	`IP_ESP_PING_LISTEN` fan-out	strongSwan plugin
send response	pinned via `ESP_PING_SEND_SPI` cmsg	strongSwan plugin

strongSwan Charon's job is narrow: **receive request → send response** only.

strongSwan/Charon responder plugin

New charon plugin `esp_ping`, raw-socket-in-a-plugin pattern.

Two sockets, deliberately separate:

- `listen_fd` — `bind()`ed, `IP_ESP_PING_LISTEN`, only ever `recvmsg()`
- `send_fd` — never listens, never sticky `IP_ESP_PING_SPI`, only opened if `respond = yes`

```
reply_spi = R-flag set ? return_spi : not supported in PoC yet
```

Deferred: validate `reply_spi` against SADB before replying, in swan plugin, not in Kernel/XFRM.

Open questions

1. **Kernel CONFIG_ESP_PING?** option with dependency on `CONFIG_XFRM_IPTFS` 1.1 **Global Netlink Enable/Disable?** seperte for Encrypted ping, or same as plain ESP ping?
2. **Rate limiting** — one global bucket for ICMP, plain ESP Ping and encrypted ESP ping.
   - separate buckets for Encrypted ESP and ICMP ping??
   - Likely /proc setting would be rejected by netdev.
   - a Netlink option would be better, but do we need it?
3. **Kernel-side auto-responder for encrypted ping** — to reply without strongswan, useful for testing.
   - Waiting on IP-TFS CC payload implementation.
   - Chris needs the same feature? Can't use Charon/Pluto(IKEd)
   - It must live in the datapath for one-way latency measurement.
   - Asymetric response would still need IKEd(strongswan)
4. **32 seq number and id?** — could be useful for long lived SA.
   - Sequence number 64 is it enough?
   - If seq is 32bit use 32 bi id too.

References

github:antonyantony linux strongswan busybox iproute2