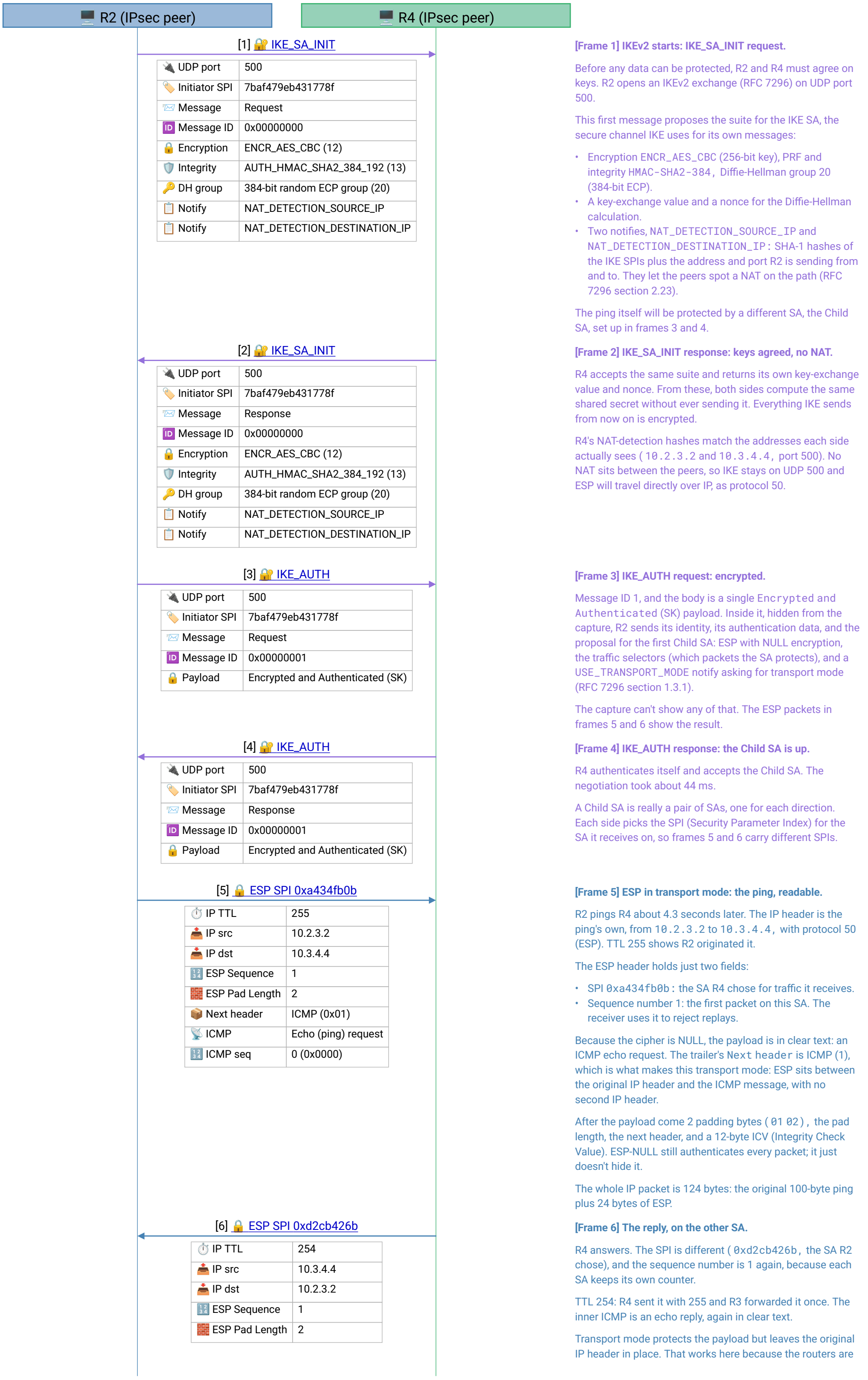
ipsec-esp-transport-mode.annotated




**[Frame 1] IKEv2 starts: IKE_SA_INIT request.**

Before any data can be protected, R2 and R4 must agree on keys. R2 opens an IKEv2 exchange (RFC 7296) on UDP port 500.

This first message proposes the suite for the IKE SA, the secure channel IKE uses for its own messages:

- Encryption `ENCR_AES_CBC` (256-bit key), PRF and integrity `HMAC-SHA2-384`, Diffie-Hellman group 20 (384-bit ECP).
- A key-exchange value and a nonce for the Diffie-Hellman calculation.
- Two notifies, `NAT_DETECTION_SOURCE_IP` and `NAT_DETECTION_DESTINATION_IP`: SHA-1 hashes of the IKE SPIs plus the address and port R2 is sending from and to. They let the peers spot a NAT on the path (RFC 7296 section 2.23).

The ping itself will be protected by a different SA, the Child SA, set up in frames 3 and 4.

**[Frame 2] IKE_SA_INIT response: keys agreed, no NAT.**

R4 accepts the same suite and returns its own key-exchange value and nonce. From these, both sides compute the same shared secret without ever sending it. Everything IKE sends from now on is encrypted.

R4's NAT-detection hashes match the addresses each side actually sees (`10.2.3.2` and `10.3.4.4`, port 500). No NAT sits between the peers, so IKE stays on UDP 500 and ESP will travel directly over IP, as protocol 50.

**[Frame 3] IKE_AUTH request: encrypted.**

Message ID 1, and the body is a single `Encrypted and Authenticated` (SK) payload. Inside it, hidden from the capture, R2 sends its identity, its authentication data, and the proposal for the first Child SA: ESP with NULL encryption, the traffic selectors (which packets the SA protects), and a `USE_TRANSPORT_MODE` notify asking for transport mode (RFC 7296 section 1.3.1).

The capture can't show any of that. The ESP packets in frames 5 and 6 show the result.

**[Frame 4] IKE_AUTH response: the Child SA is up.**

R4 authenticates itself and accepts the Child SA. The negotiation took about 44 ms.

A Child SA is really a pair of SAs, one for each direction. Each side picks the SPI (Security Parameter Index) for the SA it receives on, so frames 5 and 6 carry different SPIs.

**[Frame 5] ESP in transport mode: the ping, readable.**

R2 pings R4 about 4.3 seconds later. The IP header is the ping's own, from `10.2.3.2` to `10.3.4.4`, with protocol 50 (ESP). TTL 255 shows R2 originated it.

The ESP header holds just two fields:

- SPI `0xa434fb0b`: the SA R4 chose for traffic it receives.
- Sequence number 1: the first packet on this SA. The receiver uses it to reject replays.

Because the cipher is NULL, the payload is in clear text: an ICMP echo request. The trailer's `Next header` is ICMP (1), which is what makes this transport mode: ESP sits between the original IP header and the ICMP message, with no second IP header.

After the payload come 2 padding bytes (`01 02`), the pad length, the next header, and a 12-byte ICV (Integrity Check Value). ESP-NULL still authenticates every packet; it just doesn't hide it.

The whole IP packet is 124 bytes: the original 100-byte ping plus 24 bytes of ESP.

**[Frame 6] The reply, on the other SA.**

R4 answers. The SPI is different (`0xd2cb426b`, the SA R2 chose), and the sequence number is 1 again, because each SA keeps its own counter.

TTL 254: R4 sent it with 255 and R3 forwarded it once. The inner ICMP is an echo reply, again in clear text.

Transport mode protects the payload but leaves the original IP header in place. That works here because the routers are

🖥️ R2 (IPsec peer)

🖥️ R4 (IPsec peer)

| | |
|---|---|
| 📦 Next header | ICMP (0x01) |
| 📡 ICMP | Echo (ping) reply |
| 🔢 ICMP seq | 0 (0x0000) |

protecting their own traffic. To protect traffic between two networks, see the tunnel-mode flow.