XFRM Migrate Mark Blues

Should/Can we fix them?

Antony Antony

July 16, 2026

Linux `mark` (fwmark)

32-bit per-skb metadata (`skb->mark`) — kernel-local, not on the wire.

- **Set by**: `iptables`/`nft ... -j MARK --set-mark`, or `SO_MARK` sockopt on a socket
- **Value + mask**: `ip rule`/nft match only the masked bits — `fwmark VALUE/MASK` — so one mark can pack several fields into different bit ranges
- **Routing**: `ip rule add fwmark VALUE/MASK lookup <table>` — policy routing picks a table (route/SA) by mark
- **`ping -m`**: `ping -m N <dest>` sets `SO_MARK` on ping's socket → outgoing packets carry mark `N` → matched by the rule above

mark: setting it and routing on it

```
# iptables
iptables -t mangle -A OUTPUT -p icmp -j MARK --set-mark 0x100

# nft
nft add rule inet mangle OUTPUT meta mark set 0x100

# routing + ping
ip rule add fwmark 0x100/0xff00 lookup 100
ping -m 0x100 8.8.8.8

# XFRM/ESP

ip xfrm state add ... mark 0x100 mask 0x0
```

ip xfrm state: two SAs, same SPI, different mark

Identical src/dst/proto/spi/reqid — only mark/mask differ:

```
ip xfrm state add src 10.1.1.1 dst 10.1.1.2 proto esp spi 0x1000 \
  reqid 100 mode tunnel aead "rfc4106(gcm(aes))" 0x1111...1111 96 \
  mark 1 mask 1

ip xfrm state add src 10.1.1.1 dst 10.1.1.2 proto esp spi 0x1000 \
  reqid 100 mode tunnel aead "rfc4106(gcm(aes))" 0x1111...1111 96 \
  mark 0 mask 0
```

`ip xfrm state` lists both — only the `mark 1 mask 1` SA prints a `mark 0x1/0x1` line; the `mark 0 mask 0` SA prints no mark line at all.

Delete summary

```
# ip xfrm state add ... spi 0x1000 mark 1 mask 1 (SA_target)
# ip xfrm state add ... spi 0x1000 mark 0 mask 0
    (SA_decoy, catch-all, added after -> bucket head)
# ip xfrm state delete dst ... proto esp spi 0x1000 mark 1 mask 1
    -> deletes SA_decoy; SA_target survives, untouched
```

ip xfrm state delete: deletes the wrong SA

```
ip xfrm state delete dst 10.1.1.2 proto esp spi 0x1000 mark 1 mask 1

ip xfrm state show
  src 10.1.1.1 dst 10.1.1.2
    proto esp spi 0x00001000 reqid 100 mode tunnel
    mark 0x1/0x1
```

Deleted the `mark 0 mask 0` SA, not the requested `mark 1 mask 1` one — the surviving SA is the one with `mark 0x1/0x1`, i.e. delete matched on `spi` alone and ignored `mark`/`mask`.

MIGRATE_STATE: partial shadow — setup

A masked SA and a catch-all SA, same SPI:

```
ip xfrm state add src 10.1.1.1 dst 10.1.1.2 proto esp spi 0x1000 \
  reqid 100 mode tunnel aead "rfc4106(gcm(aes))" 0x1111...1111 96 \
  mark 0x100 mask 0xff00

ip xfrm state add src 10.1.1.1 dst 10.1.1.2 proto esp spi 0x1000 \
  reqid 100 mode tunnel aead "rfc4106(gcm(aes))" 0x1111...1111 96 \
  mark 0 mask 0
```

`ip xfrm state` lists both — one prints `mark 0x100/0xff00`, the catch-all prints no mark line.

MIGRATE_STATE: partial shadow — migrate fails

Migrate the catch-all SA to a new, seemingly unrelated mark:

```
ip xfrm state migrate dst 10.1.1.2 proto esp spi 0x1000 \
  mark 0 mask 0 new-dst 10.1.1.2 new-src 10.1.1.1 \
  new-reqid 100 new-mark 0x1ab mask 0xffffffff
```

```
Error: New SA tuple already occupied.
```

`0x1ab & 0xff00 == 0x100` — the new mark falls inside the range the `mark 0x100/0xff00` SA already claims, even though the new SA's own mask (`0xffffffff`) is exact. A wider-masked SA partially shadows narrower ones it overlaps with.

Why allow identical SAs differing only by mark?

Same sequence-number space, same crypto keys — only `mark` differs:

- Would break **replay protection** — one shared replay window, output SN wil go up and down and fail replay checks
- **AES-GCM IV reuse risk** — same key + same SN space used on two "separate" SAs
- Would violate **NIST crypto operation limits** — one key's usage budget silently spent by two paths
- What is the use case?
- Is this just misconfiguration — or should the kernel refuse it outright add?

Why fix? Sashiko/AI review triggered XFRM_MIGRATE_STATE

Sashiko review URL

https://sashiko.dev/#/patchset/20260612074725.1760473-8-steffen.klassert@secunet.com

- Should prohibit identical SAs differing only by mark?
- xfrm_state_add() and xfrm_state_update()?

Waiting: send fixes v2

- more wildcard-mark checks, reported by Sashiko and Yan
- Control-plane lookups now require exact mark/mask match
- `preflight-checks-failing: True` — must resolve before sendww
- recipient list needs refreshing
- partial shadowing still an issue. I will send v2 today