Diet-ESP for 5G Fronthaul

Fast, Lean, and Sufficient

SecPRI: Securing eCPRI with Minimal Overhead

Intel i7-5600U • AES-NI • No QAT • No Kernel Bypass

5G NR Fronthaul Packet Rate

NR Numerology 0 (μ=0, SCS = 15 kHz):

- 1 slot = 1 ms → **1,000 slots/s**
- 14 OFDM symbols per slot
- 1 eCPRI packet per symbol per antenna

Configuration	Antennas	PPS
20 MHz, 1 antenna	1	14,000
20 MHz, 4 antennas	4	**56,000**
100 MHz (μ=1), 4 ant.	4	112,000

56,000 pps = the baseline 4-antenna target for this work.

The Problem

- O-RAN fronthaul carries eCPRI between O-DU and O-RU
- Strict latency: **≤100 µs** one-way
- High packet rate: **56,000 pps** (NR µ=0, 4 antennas)
- Standard IPsec ESP adds **20 extra bytes** per packet (IV, UDP, padding)

Diet-ESP: What We Remove

Field	Standard ESP	Diet-ESP
IV (8B)	✓	✗ (implicit, RFC 8750)
UDP header (8B)	✓	✗ (fixed ports, compressed)
Padding + pad_len (2B)	✓	✗ (1-byte alignment)
Next Header (2B)	✓	✗ (always UDP, implicit)

Result: 52B overhead (IPv6) vs 72B for standard ESP transport

Implementations Tested

Variant	Language	Crypto	Lines of Code
secpri-c OpenSSL	C	AES-NI (in-process)	766
secpri-c AF_ALG	C	Kernel crypto API	766
secpri-py	Python	cryptography lib	428
esp-xfrm	Python	Kernel xfrm (standard ESP)	102

All use raw AF_PACKET sockets for packet I/O (no Scapy).

Benchmark Design

- **Sender**: tcpreplay at precise PPS (or Python raw socket for jumbo)
- **Receiver**: C program, AF_PACKET + poll(), CLOCK_MONOTONIC
- **Topology**: 4 network namespaces, veth pairs, MTU 9000
- **Max PPS**: binary search (double until loss, then bisect)
- **Latency**: fixed 14,000 pps, measure inter-packet time
- **CPU**: Intel Core i7-5600U @ 2.60 GHz (2 cores / 4 threads, 2015)
- **Crypto**: AES-NI hardware instructions (no QAT, no dedicated accelerator)

Scapy was 40× slower — removed from all data paths.

Max Sustained PPS

Max PPS: Key Numbers

Variant	8B	1200B	2048B
secpri-c OpenSSL	**60,006**	**60,006**	**60,006**
secpri-c AF_ALG	60,006	44,031	42,500
secpri-py	60,006	44,031	40,000
esp-xfrm tunnel	60,006	24,062	23,750
esp-xfrm transport	56,012	15,076	20,000

Target: 56,000 pps — only secpri-c OpenSSL meets it at all sizes.

Latency at 14,000 pps (Single Antenna)

Latency: All Variants Meet O-RAN Budget

Variant	8B	1200B	2048B
secpri-c AF_ALG	21.7 µs	49.6 µs	79.3 µs
secpri-c OpenSSL	29.6 µs	52.0 µs	83.1 µs
secpri-py	37.1 µs	42.0 µs	85.1 µs
esp-xfrm transport	70.4 µs	40.1 µs	73.7 µs

All below **100 µs** — at sub-saturation rates, every variant works.

Why Direct Implementation Wins

secpri-c OpenSSL is 2.5–4× faster than kernel xfrm because:

1. No protocol adaptation (eCPRI → ESP in one loop, not via UDP)
2. No SPD/SAD lookup (single hardcoded SA)
3. No extra syscalls for crypto (OpenSSL EVP is in-process)
4. Simplified format (no IV/padding construction)

Cost: 766 LoC vs 102 LoC — but 4× the throughput.

Why AF_ALG Degrades

Each packet requires **4 extra syscalls** for crypto:

```
accept() → sendmsg(plaintext) → read(ciphertext) → close()
```

Plus kernel↔️userspace copies grow with payload size.

At 2048B: 42,500 pps (AF_ALG) vs 60,006 pps (OpenSSL)

Scapy: The Hidden Bottleneck

Initial measurements showed **all variants at ~130 pps**.

Cause: Scapy's `sniff()` / `sendp()` in encap/decap.

Fix: raw AF_PACKET sockets → **40× improvement**.

Lesson: always profile the I/O path, not just crypto.

Conclusions

1. **Diet-ESP saves 20 bytes/packet** — no performance penalty
2. **secpri-c OpenSSL sustains 60k pps** at all sizes on a laptop CPU
3. **Kernel IPsec (xfrm) cannot meet fronthaul rates** (15k–32k pps)
4. **Packet I/O dominates over crypto** — socket choice matters more than algorithm
5. **All variants meet 100 µs latency** at per-antenna rates
6. **For 4-antenna aggregation, only the C implementation suffices**

What's Next

- Larger payloads (4800B, 8000B) with jumbo sender
- QAT hardware offload (secpri-qatlib)
- AF_XDP kernel bypass for > 100k pps
- Multi-core (RSS / SO_REUSEPORT)
- Real NIC measurements (not veth)