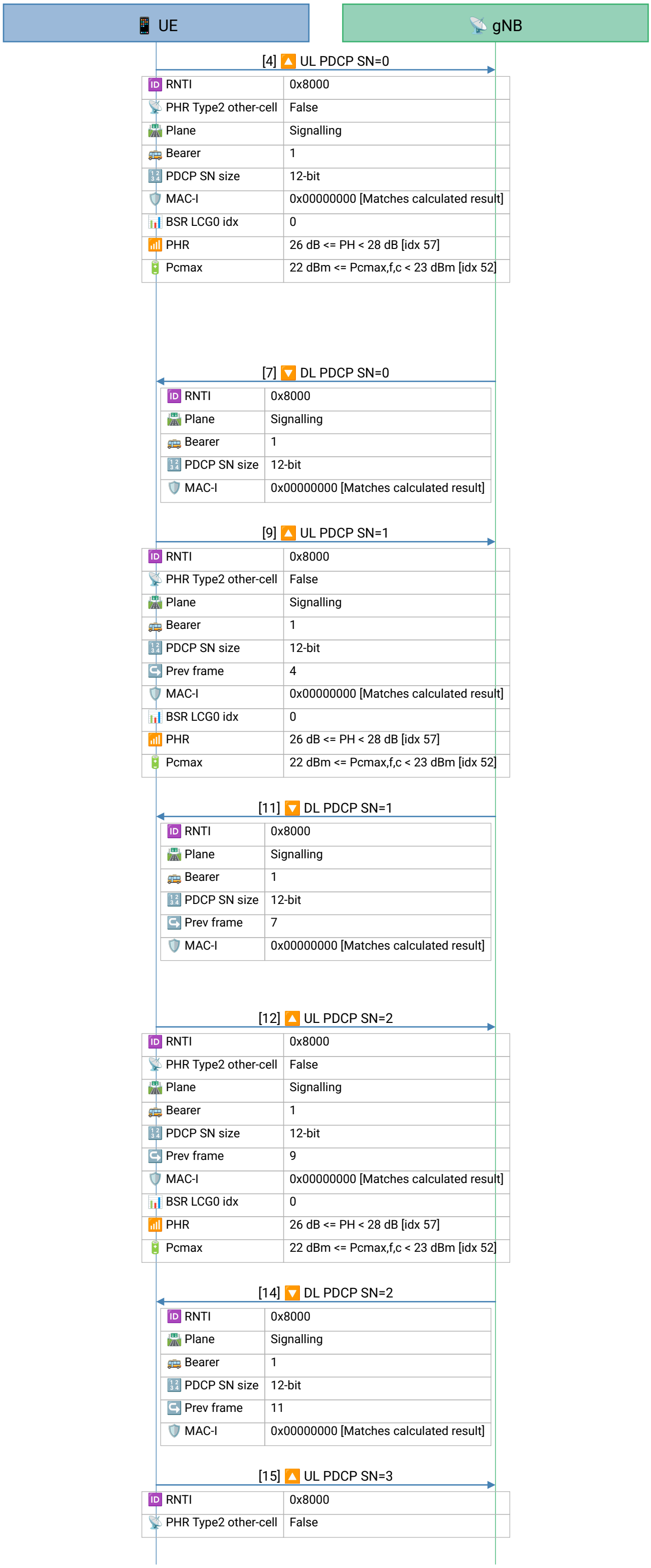
pdcp-nr-sequence-numbers.annotated



[Frame 4] The first PDCP PDU of the capture, and the whole SRB header format in two bytes. The header is `0000 .... = Reserved: 0` followed by `.... 0000 0000 0000 = Seq Num: 0` -- four reserved bits and a 12-bit sequence number. There is no D/C (Data/Control) bit here: the signaling-radio-bearer PDU format has no way to express a PDCP Control PDU, because SRBs never carry one (TS 38.323 §6.2.2.1). Behind the SDU sits a four-byte MAC-I trailer, and Wireshark reads it as `MAC: 0x00000000 [Matches calculated result]`. That is not a broken digest. AS security has not been activated yet, and TS 38.323 §6.3.4 says the field is still present on an SRB and padded with zeros until it is. Note also what is *already* dark: the SDU is an `RRCSetupComplete` carrying a NAS Registration Request, and inside that request the NAS `message container` IE reads `Encrypted data`. The NAS layer runs its own security, on its own keys, from long before PDCP does.

[Frame 7] The downlink has its own counter. The gNB's first PDCP PDU on this bearer is also `Seq Num: 0`. It is not a repeat of frame 4. Each end runs its own PDCP entity and each keeps its own `TX_NEXT` for what it transmits (TS 38.323 §7.1), so uplink SN 0 and downlink SN 0 were minted by different entities and are unrelated packets. Same zeroed MAC-I, for the same reason as frame 4.

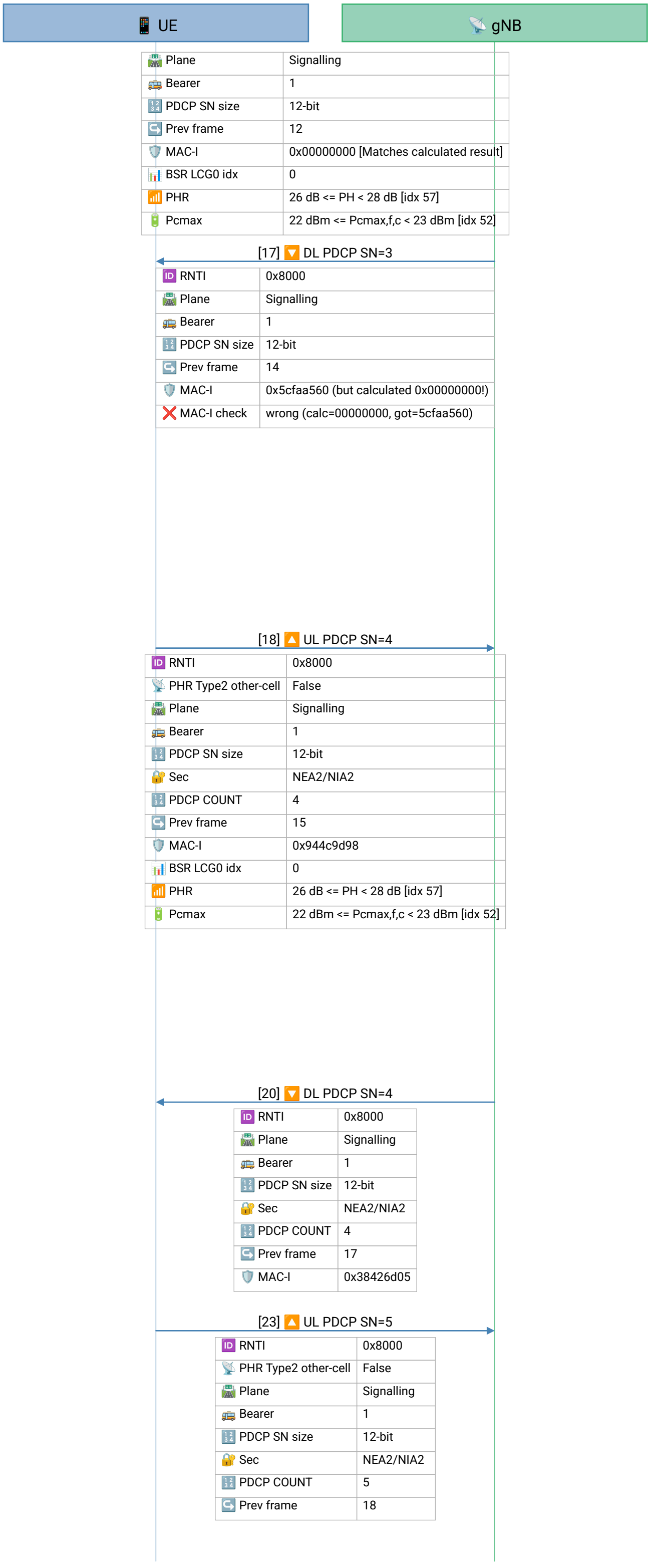
[Frame 9] SN 1, and the counter's only job. Nothing about this PDU is special, which is the point: the sequence number increments by one per PDCP SDU submitted for transmission and never skips. Downstream, RLC will number, segment and retransmit these bytes on its own schedule; the PDCP number stays attached to the *message*. Wireshark agrees the ordering is clean -- `Sequence Analysis - OK, Expected SN: 1.`

[Frame 11] Downlink SN 1 -- and the other security context announcing itself. The RRC wrapper is a `DLInformationTransfer`; the NAS message inside it is a NAS Security Mode Command, sent with security header type `Integrity protected with new 5GS security context (3)`, NAS sequence number 0 and message authentication code `0x41bf0b2b`. It selects `128-5G-EA2` for ciphering and `128-5G-IA2` for integrity. None of that is PDCP's business: this PDCP PDU is still unprotected, with a zeroed MAC-I. Two security layers, negotiated separately, on the same bearer. The conceptual account of that split lives in the companion article on decrypting this capture.

[Frame 12] Uplink SN 2 -- the clearest single frame in the capture on what PDCP does and does not hide. At the PDCP layer this PDU is wide open: no ciphering yet, MAC-I all zeros. At the NAS layer it is completely opaque -- security header type `Integrity protected and ciphered with new 5GS security context (4)`, MAC `0x97f62856`, and then `Encrypted data`. An unciphered PDCP PDU whose payload is still unreadable. The two contexts genuinely are independent.

[Frame 14] Downlink SN 2, a second NAS Security Mode Command. Same security header type as frame 11, but NAS sequence number 1 and a different message authentication code, `0x7acf026e`. Note that this NAS sequence number is counting in a space of its own: it has just gone 0, 1 while the PDCP downlink number went 1, 2. Two counters, two layers, no relationship.

[Frame 15] Uplink SN 3 -- the last unprotected PDU of the connection. Frames 4, 7, 9, 11, 12, 14 and 15 are the complete list of PDUs here whose MAC-I reads `0x00000000`. From the next PDU onwards, in both directions, every value in that trailer is a real digest.



[Frame 17] The hinge: the RRC Security Mode Command, and where AS security actually starts. The SDU is a `securityModeCommand` selecting `cipheringAlgorithm: nea2 (2)` and `integrityProtAlgorithm: nia2 (2)` -- AES for both. Two things to notice, and they are the heart of this page.

First, the MAC-I is suddenly real: `0x5cfaa560`, not zeros. Integrity protection is applied to this very message, using the new key.

Second, the message is still readable. Ciphering is *not* applied to it. TS 38.331 §5.3.4.3 draws the line exactly there -- integrity from this message onward, ciphering only after the procedure completes. The RRC decodes in full without any key being used.

Wireshark flags `MAC-I Digest wrong - calculated 00000000 but found 5cfaa560`, and that is a tooling artifact rather than a protocol error: the dissector only learns the algorithms *from* this frame, so it validated the digest before it had them. Every subsequent PDU of this connection -- on both bearers -- names this frame as its `Configuration frame: 17`.

[Frame 18] Uplink SN 4 -- the security context becomes visible, and so does COUNT. Wireshark now prints the full input set it will use for the rest of the connection:

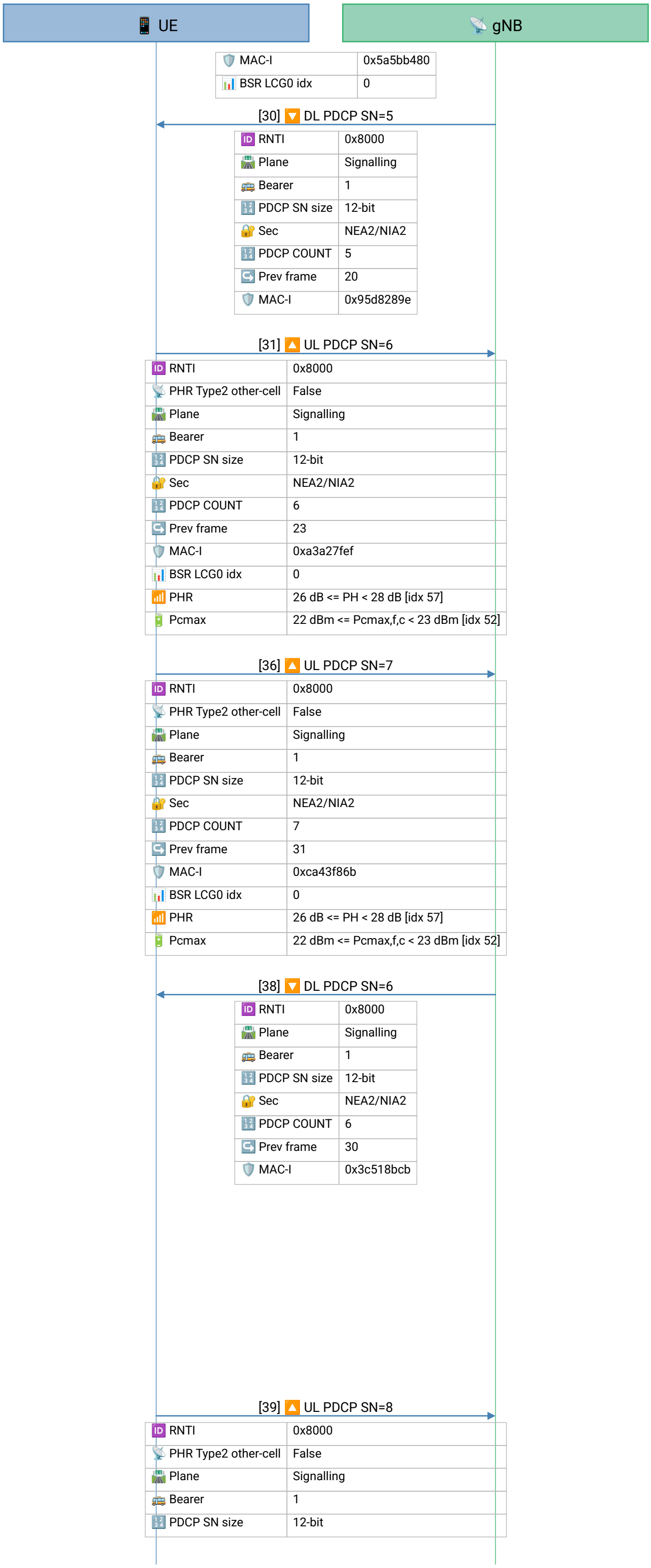
- `Ciphering Algorithm: NEA2 (AES) (2)`
- `Integrity Algorithm: NIA2 (AES) (2)`
- `BEARER: 0`
- `DIRECTION: Uplink (0)`
- `COUNT: 4`
- `CIPHER KEY: 95F23667A60D91689B31772783E6FDDD`

`COUNT: 4` is the number to look at. COUNT is 32 bits: a Hyper Frame Number concatenated with the sequence number (TS 38.323). Only the sequence number travels on the wire; the HFN is state each end keeps and increments when the SN wraps. Here the SN has not wrapped, so COUNT and SN read the same -- 4 and 4 -- and they do so on every single PDU in this session. `BEARER: 0` is the bearer input to the cipher, and `DIRECTION` is one bit. Change any one of those four and you get a different keystream, which is precisely why uplink SN 4 and downlink SN 4 are safe to cipher under the same key.

This message, the `securityModeComplete`, is itself integrity protected -- MAC-I `0x944c9d98` -- but still not ciphered, the mirror of frame 17. Wireshark did not need to decipher it.

[Frame 20] The first ciphertext on the bearer. Downlink SN 4 is the first PDU in this capture that Wireshark had to run through AES to read, and the trace says so: a `Deciphered Data` field appears for the first time, holding `300201200100100038426d05`. Look at how it ends -- `38426d05` -- and compare it with the frame's `MAC: 0x38426d05`. The MAC-I is inside the ciphered region. TS 38.323 §5.8 puts both the data part and the MAC-I under the cipher, so an eavesdropper cannot even see the integrity tag, let alone the payload. The SDU is a `UECapabilityEnquiry`; it is readable here only because the capture ships with the AS keys.

[Frame 23] One sequence number, 1667 bytes, three RLC transmissions. Uplink SN 5 carries the `UECapabilityInformation` -- the largest message on this bearer, and Wireshark reassembles it to 1667 bytes. It occupies exactly one PDCP sequence number. PDCP numbers the SDU it was handed; how many pieces the scheduler forced RLC to cut it into is invisible from up here. That is the cleanest statement of what the layer boundary buys you, and it is the same event the RLC companion walks as its three-segment set piece across frames 21 to 23.



[Frame 30] Downlink SN 5 -- ciphered at PDCP, and still ciphered underneath. The `DLInformationTransfer` decodes because we hold the AS keys. The NAS message inside it does not: security header type `Integrity protected and ciphered (2)`, message authentication code `0x6b39d3f8`, NAS sequence number 2, then `Encrypted data`. Deciphering PDCP peels off exactly one layer.

[Frame 31] Uplink SN 6 -- two sequence numbers and two integrity tags in one packet. PDCP contributes `Seq Num: 6`, `COUNT: 6` and `MAC: 0xa3a27fef`. The NAS message inside contributes its own sequence number, 2, and its own message authentication code, `0x7bc1bd11`. Neither counter knows the other exists. If you are ever tempted to correlate a NAS sequence number with a PDCP one, this frame is the counterexample.

[Frame 36] Uplink SN 7 after a 1368.993 ms pause, and the counter does not care. The TX_NEXT state variable is not a timer; it simply holds whatever value it reached. NAS sequence number 3 inside, still `Encrypted data`.

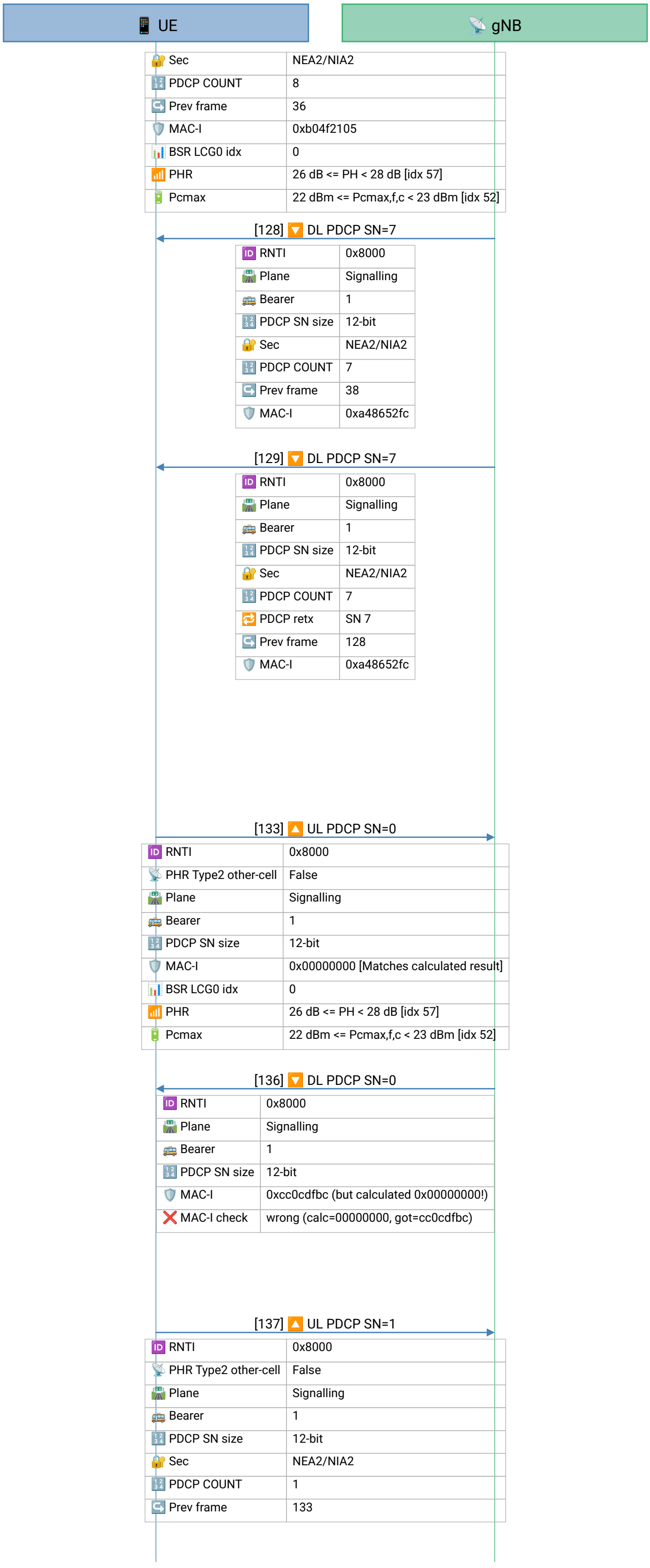
[Frame 38] Downlink SN 6 -- the frame that creates the second PDCP entity. The SDU is an `RRCReconfiguration`, and its `drb-ToAddModList` adds DRB 1 with a `pdcp-Config` that determines everything the second diagram shows:

- `pdcp-SN-SizeUL: len18bits (1)` and `pdcp-SN-SizeDL: len18bits (1)`
- `discardTimer: infinity (15)`
- `integrityProtection` -- NOT present
- `headerCompression: notUsed (0)`
- `statusReportRequired: true (0)`
- `t-Reordering` -- NOT present

`integrityProtection is NOT present` is the one to remember: it is why every PDCP PDU on the data bearer arrives with `MAC-I Present: False`, and why the data bearer's PDUs are ciphered but not integrity protected.

The same message adds SRB 2 with `pdcp-Config is NOT present`, and it carries an `sdap-Config` -- `sdap-HeaderUL: present (0)`, `sdap-HeaderDL: absent (1)`, QFI 2 -- which is why the SDAP header shows up on the data bearer's uplink and never on its downlink. The `securityConfig` re-states nea2 and nia2 with `keyToUse: master (0)`.

[Frame 39] Uplink SN 8, and the smallest complete illustration of the PDU layout. The `RRCReconfigurationComplete` deciphers to `0800b04f2105`: two bytes of RRC, then `b04f2105`, which is exactly what the MAC field displays as `0xb04f2105`. Header in the clear, payload and integrity tag under the cipher. This is the last PDU of the connection's opening phase -- the data bearer takes over from here, and this bearer goes quiet for more than ten seconds.



[Frame 128] Ten seconds later, downlink SN 7 continues from the SN 6 of frame 38. The entity kept its full state through the silence -- no keepalives, no resets, just the next number. The SDU deciphers to `1000a48652fc`: an eight-byte PDU carrying an RRC Release and the MAC-I `0xa48652fc`. Remember those bytes.

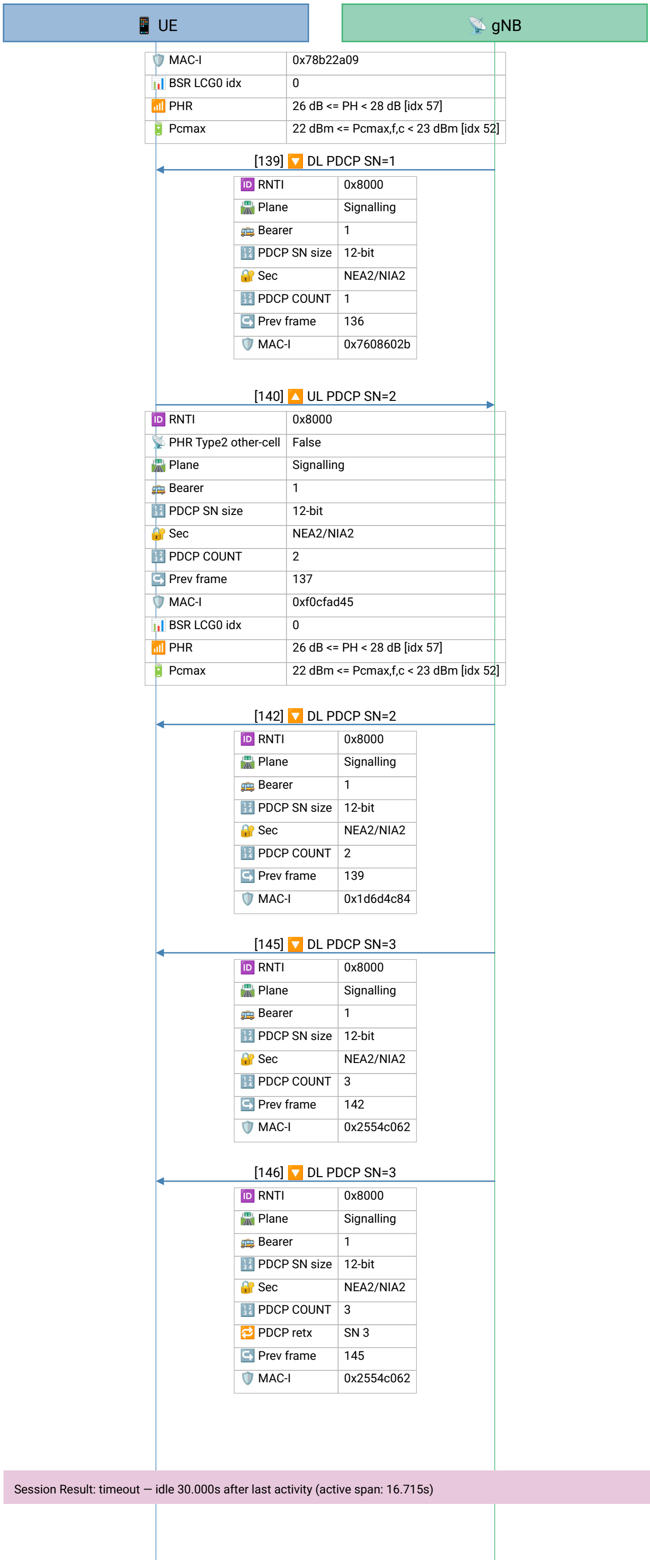
[Frame 129] The same PDU again, 44.894 ms later, byte for byte. Wireshark heads the sequence-analysis block `Sequence Analysis- SN 7 Repeated` and raises `PDCP SN (7) repeated for Downlink for UE 1 (DCCH-1)`, with `Expected SN: 8` and `OK: False` -- the only downlink PDU of this connection where that check fails. The deciphered data is `1000a48652fc` -- identical to frame 128 -- and the MAC-I is the same `0xa48652fc`. That identity is the whole lesson. This is not PDCP re-sending anything: PDCP has no retransmission machinery. RLC's `t-PollRetransmit` expired without a status report and RLC put the *same stored PDU* back on the wire. Because PDCP had already assigned COUNT 7 and ciphered under it, the ciphertext is unchanged.

And that is the correct behavior, not a flaw. TS 33.501 forbids reusing a COUNT value with the same key, bearer and direction for *different* plaintext, because that would reuse keystream. Re-sending the identical PDU reuses nothing new. The rule an analyst needs is the practical one: a repeated PDCP SN with identical ciphertext is a lower-layer retransmission; a repeated PDCP SN with *different* ciphertext would be a genuine security incident.

[Frame 133] A new connection, and the counter starts over. Uplink `Seq Num: 0` on SRB 1 again, and Wireshark's configuration line now reads `UEId= 2` where every frame up to 129 read `UEId= 1`. A PDCP entity lives and dies with its bearer, so `TX_NEXT`, the COUNT state and the security context all start from nothing. The MAC-I is `0x00000000 [Matches calculated result]` once more, and there is no `Previous frame for Bearer` link back to frame 129 -- as far as PDCP is concerned this is a bearer one frame old. Everything frames 4 to 39 taught applies again, unchanged.

[Frame 136] The second Security Mode Command, and the experiment that proves frame 17. Downlink SN 0 carries an RRC `securityModeCommand` with a real MAC-I, `0xcc0cdfbc`, and Wireshark raises the same `MAC-I Digest wrong - calculated 00000000 but found cc0cdfbc` it raised on frame 17. Every frame after this one names `Configuration frame: 136`. Here is what makes this connection interesting: the capture's key table only holds keys for UEId 1. We have no keys at all for this UE context. And yet this message still decodes completely -- because, exactly as TS 38.331 §5.3.4.3 says, a Security Mode Command is integrity protected but not ciphered. Nothing was deciphered to read it.

[Frame 137] Uplink SN 1, the Security Mode Complete -- still readable, still without keys. MAC-I `0x78b22a09`, a real digest. No `Deciphered Data` field, because none was needed. This is the mirror of frame 18, and the second half of the proof: on a connection we cannot decrypt, the last two messages in the clear are the two that bracket security activation.



[Frame 139] And now it goes dark. Downlink SN 1 is the first PDU after the Security Mode procedure on this connection, and where frames 133, 136 and 137 showed RRC, Wireshark can only offer `Signalling Data: 38a4bc1a51d9573b91b80215131c5150....` No keys, no plaintext. Compare frame 20, which was the first ciphered PDU of the first connection and *was* readable: the boundary sits in exactly the same place on both connections. Only our ability to see past it differs.

The PDCP header is still perfectly legible, of course -- `Seq Num: 1`, the 12-bit format, the MAC-I `0x7608602b`. Ciphering starts *below* the header, which is what lets a gNB route and reorder PDUs it has not yet deciphered.

[Frame 140] Uplink SN 2 -- two opaque bytes. `Signalling Data: 1f6d`, and MAC-I `0xf0cfad45`. Even a two-byte payload keeps its four-byte integrity tag: on an SRB the MAC-I is mandatory regardless of how little there is to protect.

[Frame 142] Downlink SN 2, opaque. `Signalling Data: 5c03af27593b87a53db1039d0f41a11bb3635315fe`, MAC-I `0x1d6d4c84`. The sequence numbers keep marching in the clear while the content stays hidden -- header-level analysis of a bearer you have no keys for is still perfectly possible.

[Frame 145] Downlink SN 3, the last new PDU of the capture. `Signalling Data: b109`, MAC-I `0x2554c062`. Two bytes of ciphertext and four bytes of digest.

[Frame 146] The second timer-driven retransmission, and it confirms frame 129 from the other side of the key boundary. Wireshark reports `PDCP SN (3) repeated for Downlink for UE 2 (DCCH-1), repeated-frame: True, OK: False`, 44.915 ms after frame 145. The signaling data is `b109` and the MAC-I is `0x2554c062` -- byte-identical to frame 145, exactly as frames 128 and 129 were byte-identical to each other.

That is the same phenomenon observed twice, on two different connections, with and without keys: RLC re-sends a stored PDU, PDCP's COUNT does not advance, and the bytes on the wire repeat. You did not need to decrypt anything to diagnose it. This SRB session ends here, mid-retransmission. The capture runs on: 32.45 s later a third connection sets up, and frame 150 carries its NAS Service Request at PDCP SN `0` — a new entity, counting from zero again.

Session Result: timeout — idle 30.000s after last activity (active span: 16.715s)