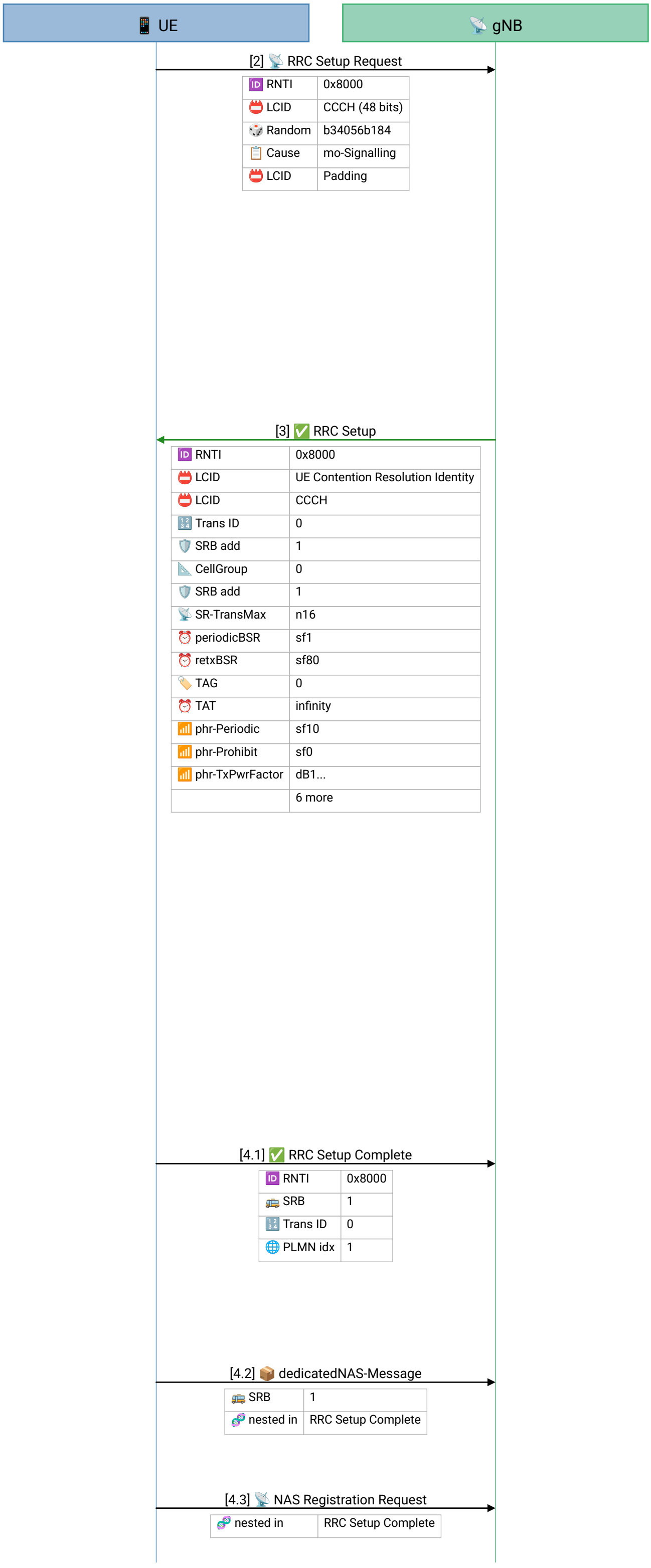


[Frame 2] Msg3 — the UE names itself before it has a name. This is a `UL-CCCH-Message`: the Common Control Channel, the only channel that exists before a dedicated bearer does. There is no `rrc-TransactionIdentifier` on an `RRCSetupRequest` — a transaction identifier exists to pair a response with a command, and nothing here has been commanded yet.

Three fields, and that is the whole message:

- `ue-Identity` is a CHOICE, and this connection takes `randomValue (1)`: 39 bits, `b34056b184`. The UE draws a random number because the NAS layer gave RRC no subscriber identity to use.
- `establishmentCause` is `mo-Signalling (3)` — mobile-originated signaling. The UE wants to talk to the core, not to move bulk data.
- `spare` — one bit, zero, padding the message out to the fixed CCCH size.

Remember the identity choice. The second connection, at frame 131, makes the other one, and the difference is the most informative thing on this whole timeline.

[Frame 3] Msg4 — the RRC Setup, 0.708 ms later. A `DL-CCCH-Message`, still on the common channel because the UE has no dedicated bearer yet. `rrc-TransactionIdentifier: 0` opens the first RRC transaction of the connection.

The message has exactly two payload IEs, and they are wildly different sizes:

- `radioBearerConfig` holds a one-item `srb-ToAddModList` naming `srb-Identity: 1`. That is all. No `pdcp-Config` — so SRB 1 takes the specification's default PDCP profile rather than anything signaled here.
- `masterCellGroup` is a nested, separately encoded OCTET STRING of **209 octets,** and everything else lives inside it.

Unpacked, that 209-octet blob is a `CellGroupConfig` with `cellGroupId: 0` and four sections. Its `rlc-BearerToAddModList` carries one `RLC-BearerConfig: logicalChannelIdentity: 1, servedRadioBearer: srb-Identity 1`, and a `mac-LogicalChannelConfig` with `priority: 1, prioritisedBitRate: infinity, bucketSizeDuration: ms50, logicalChannelGroup: 0`. Note what is *missing* from it: there is no `rlc-Config`. SRB 1's RLC entity is never configured anywhere in this capture — it runs the TS 38.331 default acknowledged-mode profile.

`mac-CellGroupConfig` sets `sr-TransMax: n16, periodicBSR-Timer: sf1, retxBSR-Timer: sf80,` one timing-advance group (`tag-Id: 0, timeAlignmentTimer: infinity`) and power-headroom reporting (`phr-PeriodicTimer: sf10, phr-ProhibitTimer: sf0, phr-Tx-PowerFactorChange: dB1`). `physicalCellGroupConfig` sets `p-NR-FR1: 23 dBm`. And `spCellConfig` carries the radio-link-failure watchdogs — `t310: ms1000, n310: n1, n311: n1, t311: ms1000` — plus the CORESET and search space the UE will monitor for its downlink grants.

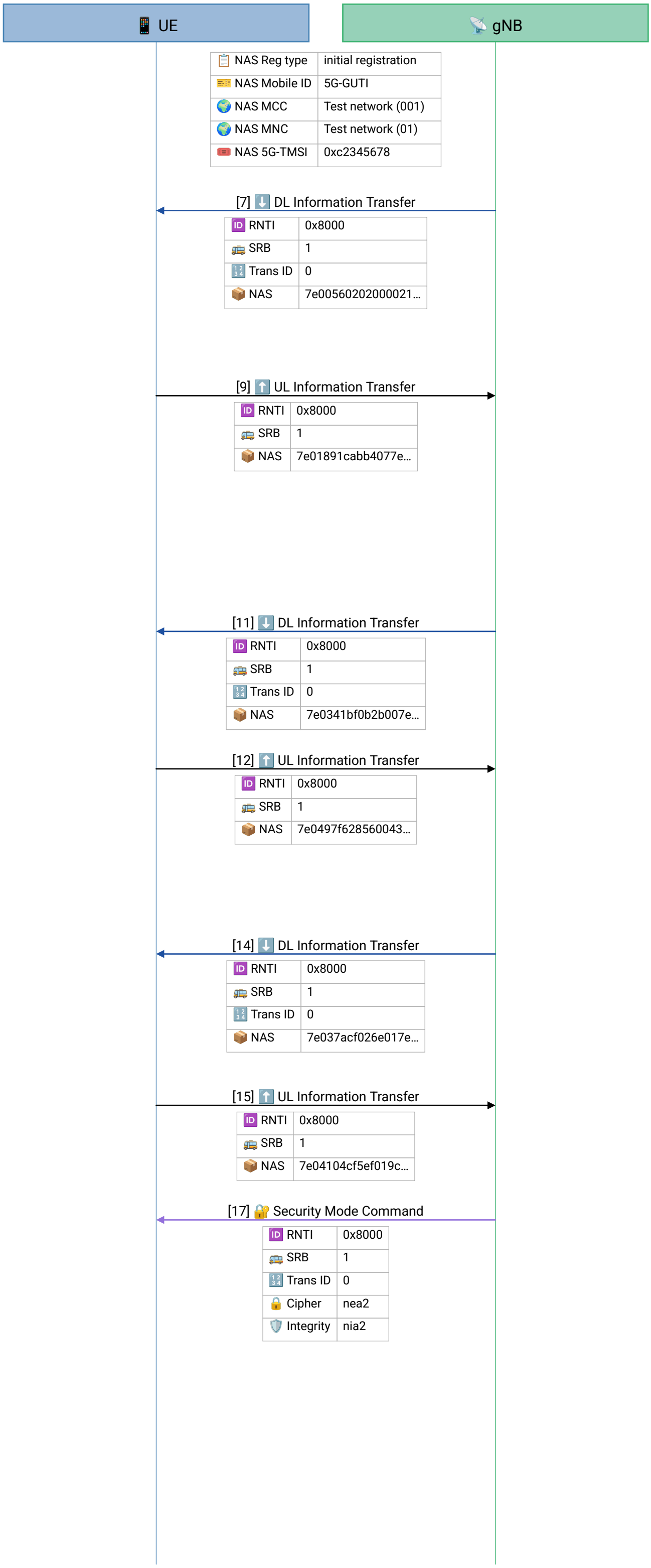
On receipt the UE enters `RRC_CONNECTED`.

[Frame 4.1] Msg5 — the first message on a dedicated bearer. The channel changes: this is a `UL-DCCH-Message` on SRB 1, not CCCH. `rrc-TransactionIdentifier: 0` closes the transaction frame 3 opened.

Two RRC-owned IEs do routing work here. `selectedPLMN-Identity: 1` tells the gNB which broadcast PLMN the UE picked. `registeredAMF` names the core-network node the UE was last registered with — `amf-Identifier: fe0041`, with `guami-Type: native` — so the gNB can hand the NAS message that follows to the right AMF instead of guessing. Frame 133 will do this differently, and for a reason worth waiting for.

[Frame 4.2] The NAS container, seen the way RRC sees it. `dedicatedNAS-Message` is an OCTET STRING of **92 octets,** and to RRC that is all it is: an opaque payload it carries and does not read. RRC is a courier here, not a participant. Every `DLInformationTransfer` and `ULInformationTransfer` later in this session is the same courier job with a thinner envelope.

[Frame 4.3] What is inside the container. Wireshark opens the 92 octets: a Registration Request with `Security header type: Integrity protected (1)`, message



authentication code `0x97a0a3f4`, NAS sequence number 6, and then a plain 5GMM body. 5GS `registration type: initial registration (1)`, mobile identity `5G-GUTI` with `5G-TMSI: 0xc2345678`. The UE therefore already holds a NAS security context from an earlier visit — an integrity-protected initial NAS message is not a first-ever power-on. Hold on to that `5G-TMSI` value: it reappears unchanged at frame 133, on the other side of a release.

[Frame 7] The courier phase begins. A `DLInformationTransfer` carrying `rrc-TransactionIdentifier: 0` and a **42** -octet `dedicatedNAS-Message`. Inside it, an Authentication Request with `Security header type: Plain NAS message, not security protected (0)` — the core has not switched its own ciphering on yet.

RRC contributes nothing to this exchange except delivery. From here to frame 36, most of what RRC does is hold the door open while the NAS layer runs its own procedures.

[Frame 9] The reply, and an asymmetry in the RRC message itself. A `ULInformationTransfer` with a **28** -octet container holding the Authentication Response. Look at what is absent: there is no `rrc-TransactionIdentifier` on a `ULInformationTransfer`, whereas the downlink one at frame 7 had one.

That is not an omission in the capture — TS 38.331 simply does not define the field for this message. Transaction identifiers belong to network-initiated procedures; a UE pushing a NAS message upward is not answering an RRC command, so there is no transaction to name. Every uplink transfer in this session — frames 9, 12, 15, 31 and 36 — is missing the field for the same reason.

[Frame 11] NAS security starts, invisibly to RRC. An **18** -octet container holding a NAS Security Mode Command with `Security header type: Integrity protected with new 5GS security context (3)`. The core is establishing its own security context, with its own keys, entirely inside a payload RRC cannot read.

[Frame 12] The moment the NAS payload goes dark. The container jumps to **74** octets and now reads `Security header type: Integrity protected` and `ciphered with new 5GS security context (4)`, followed by `Encrypted data`. From this frame on, most NAS bodies in the capture are sealed.

Nothing changed at the RRC layer. This is the first hint of the capture's central asymmetry: the radio and the core each run their own security, and the published UE keys unlock only one of them.

[Frame 14] The same NAS procedure, run a second time. A **23** -octet container, `Security header type: Integrity protected with new 5GS security context (3)` again, another NAS Security Mode Command. The NAS sequence number has advanced from 0 to 1, so this is a genuine second procedure rather than a retransmission.

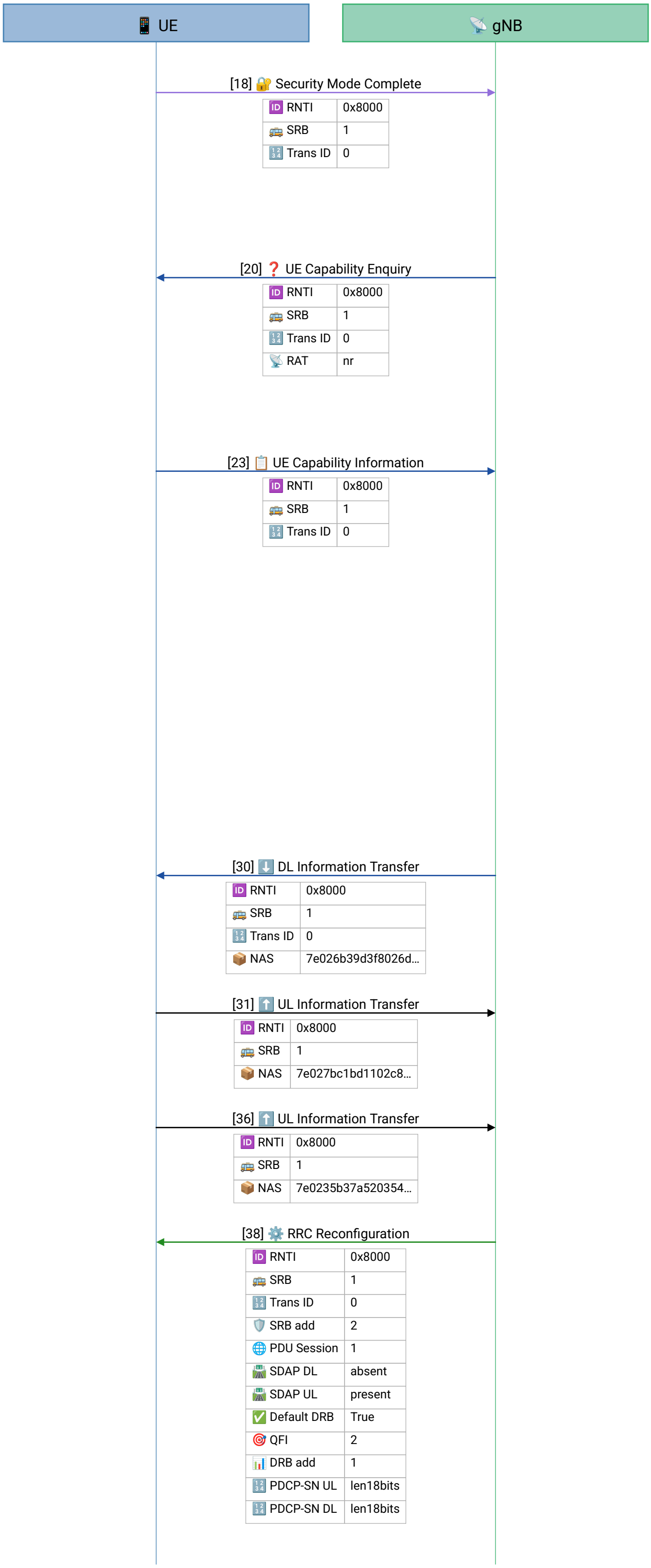
RRC's transaction identifier stays at `0` throughout, because none of this is an RRC procedure at all.

[Frame 15] And its completion. A **10** -octet container, security header type 4, ciphered. The NAS conversation is now fully protected. The radio conversation still is not — which is the next two frames' job.

[Frame 17] The AS Security Mode Command — the one security procedure that belongs to RRC. 81 bytes on the wire, `rrc-TransactionIdentifier: 0`, and a single IE: `securityConfigSMC` containing `securityAlgorithmConfig` with `cipheringAlgorithm: nea2 (2)` and `integrityProtAlgorithm: nia2 (2)`. AES-CTR for confidentiality, AES-CMAC for integrity.

Two things this message does *not* contain are as important as what it does. There is no key: both ends derive `K_RRCenc`, `K_RRCint` and `K_UPenc` locally from a key hierarchy that never crosses the air. And there is no `nas-Container`, so this is a plain security activation, not a key change on handover.

Note also what the algorithm selection covers. This one message secures *every* radio bearer of the connection — including DRB 1, which does not exist yet and will not until frame 38. The PDCP companion shows the exact frame on which ciphering actually begins; Decrypting a 5G NR radio



capture has the key hierarchy.

[Frame 18] Security Mode Complete — an empty message that matters. 10.220 ms after the command. Its entire content is `rrc-TransactionIdentifier: 0`: no IEs, no parameters, not even an acknowledgment of the algorithms it just accepted.

The message body is empty because the confirmation is not in the body — it is in the integrity tag PDCP wraps around it. If the UE had derived the wrong key, the digest would not verify and the gNB would know. An empty RRC message whose only payload is proof of a shared secret.

[Frame 20] UE Capability Enquiry — and it is a narrow question. `rrc-TransactionIdentifier: 0`, one `UE-CapabilityRAT-Request` with `rat-Type: nr`. What makes it interesting is the optional `capabilityRequestFilter`, four octets, `80040040`, which decodes to a `frequencyBandListFilter` with one entry: `bandNR: 5`. The gNB is not asking "what can you do?" — it is asking "what can you do on NR band n5?". Without that filter a modern UE would answer with a report several times larger. The answer is still the biggest message in the capture.

[Frame 23] UE Capability Information — the reply, and the reason RLC had to work. `rrc-TransactionIdentifier: 0` matches the enquiry, and one `UE-CapabilityRAT-Container` of **1656** octets holds everything else.

Inside, `accessStratumRelease: rel16`. Then per-layer capability blocks that read like an inventory of the rest of this article series: `pdcp-Parameters` (ROHC profiles 0x0000, 0x0001 and 0x0002 supported, `maxNumberROHC-ContextSessions: cs24`, `shortSN: supported`), `rlc-Parameters` (`am-WithShortSN`, `um-WithShortSN` and `um-WithLongSN` all supported), `mac-Parameters` (`longDRX-Cycle` and `shortDRX-Cycle` supported), then `phy-Parameters`, the feature-set machinery, and — for a capture named after Minimization of Drive Tests — `loggedMeasurements-r16: supported (0)`. This single container is what forces the capture's only segmentation run: the frame carrying it is 691 bytes on the wire, so RLC had to cut the message into three pieces and reassemble it. RRC never sees the split — it is handed one complete message; the RLC companion walks the same event from below.

[Frame 30] Back to couriering, now with both layers protected. A **53** -octet container, `Security header type: Integrity protected and ciphered (2)`, NAS sequence number 2. Security header type 2 rather than 4 — the "new security context" marker is gone, so the NAS context established at frames 11-15 is now simply in use.

[Frame 31] A ten-octet answer. Ciphered, security header type 2. RRC does not know or care whether this is an accept, a reject, or a completion — it is ten octets of somebody else's conversation.

[Frame 36] A 92-octet container, ciphered — the same size as the Registration Request this connection opened with. Whatever session-establishment request it carries, the network's answer arrives two frames later — and that answer is the one RRC message on this timeline that builds a data bearer.

[Frame 38] The richest message in the capture. One `RRCReconfiguration`, `rrc-TransactionIdentifier: 0`, 235 bytes on the wire, and it configures four layers at once plus carries a NAS payload. Read it in four parts.

1. The bearers it adds. `radioBearerConfig` holds `srb-ToAddModList` with `srb-Identity: 2` — a second signaling bearer, with no `pdcp-Config`, so defaults again — and `drb-ToAddModList` with `drb-Identity: 1`, the capture's only data bearer.

2. What DRB 1 gets. Its `cnAssociation` is an `sdap-Config: pdu-Session: 1`, `sdap-HeaderDL: absent (1)`, `sdap-HeaderUL: present (0)`, `defaultDRB: True`, and a one-item `mappedQoS-FlowsToAdd` naming `QFI: 2`. Those two header switches are the whole reason every SDAP PDU in this capture is an uplink one. Its `pdcp-Config` sets `discardTimer: infinity`, `pdcp-SN-SizeUL:`

📱 UE | 📡 gNB

🗜️ ROHC	notUsed
🎫 LCID	4
📊 DRB add	1...
	10 more

[39] ✅ RRC Reconfiguration Complete

🆔 RNTI	0x8000
🚌 SRB	1
🔢 Trans ID	0

[128] 🛑 RRC Release

🆔 RNTI	0x8000
🚌 SRB	1
🔢 Trans ID	0

[129] 🛑 RRC Release

🆔 RNTI	0x8000
🚌 SRB	1
🔁 RLC retx (dup PDCP SN 7)	
🔢 Trans ID	0

[131] 📡 RRC Setup Request

🆔 RNTI	0x8000
🎫 LCID	CCCH (48 bits)
📋 Cause	mt-Access
🎫 LCID	Padding

`len18bits`, `pdcp-SN-SizeDL: len18bits`, `headerCompression: notUsed`, `statusReportRequired: true`, and — worth noticing — no `integrityProtection`, so user-plane data is ciphered but not integrity-protected.

3. The security restatement. `securityConfig` repeats `cipheringAlgorithm: nea2` and `integrityProtAlgorithm: nia2` and adds `keyToUse: master`. There is no `masterKeyUpdate`, so this is not a re-key — it is the new bearers being told which existing key set to inherit.

4. The 18-octet `masterCellGroup`. Compare that with frame 3's 209 octets: a cell-group configuration is a *delta*, not a replacement. `mac-CellGroupConfig`, `physicalCellGroupConfig` and `spCellConfig` are all absent, so everything frame 3 set stays exactly as it was. What this one adds is a two-item `rlc-BearerToAddModList`: DRB 1 on `logicalChannelIdentity: 4`, `priority: 1`, `logicalChannelGroup: 1`, and a full `rlc-Config`. SRB 2 on `logicalChannelIdentity: 2`, `priority: 3`, `logicalChannelGroup: 0`, and **no** `rlc-Config` — defaults again.

DRB 1's `rlc-Config` is the only RLC configuration anywhere in this capture: `ul-AM-RLC` with `sn-FieldLength: size18`, `t-PollRetransmit: ms80`, `pollPDU: p32768`, `pollByte: kB750`, `maxRetxThreshold: t8`; `dl-AM-RLC` with `sn-FieldLength: size18`, `t-Reassembly: ms80`, `t-StatusProhibit: ms30`. And finally, `dedicatedNAS-MessageList` with one item of **122** octets — security header type 2, message authentication code `0x9b072356`, NAS sequence number 3, then `Encrypted data`. A frame in which the RRC decodes completely and the NAS payload inside it does not: the two-security-layer story in a single packet.

Also worth noting for what is *not* here: no `measConfig`, no `secondaryCellGroup`. No measurement reporting is ever configured in this trace, despite the UE advertising R16 logged-measurement support at frame 23.

[Frame 39] Reconfiguration Complete — 18.216 ms, and DRB 1 is live. Content: `rrc-TransactionIdentifier: 0`. Nothing else. Like the Security Mode Complete at frame 18, the confirmation is the message's existence, not its contents.

This is the last RRC message for a long while. Frames 40 to 127 carry no RRC at all — the connection is busy, but the traffic belongs to SDAP, PDCP, RLC and MAC. RRC built the road and then got out of the way.

[Frame 128] RRC Release, and it is as plain as an RRC Release gets. 79 bytes on the wire, `rrc-TransactionIdentifier: 0`, and every optional IE explicitly absent: no `redirectedCarrierInfo`, no `cellReselectionPriorities`, no `deprioritisationReq`, and — the important one — no `suspendConfig`. That last absence is the whole state transition. With a `suspendConfig` the UE would move to `RRC_INACTIVE`, keeping its context for a cheap resume. Without one it goes to `RRC_IDLE` and the AS context is torn down: keys, bearers, and both RLC entities cease to exist. Everything frames 3, 17 and 38 configured is discarded here.

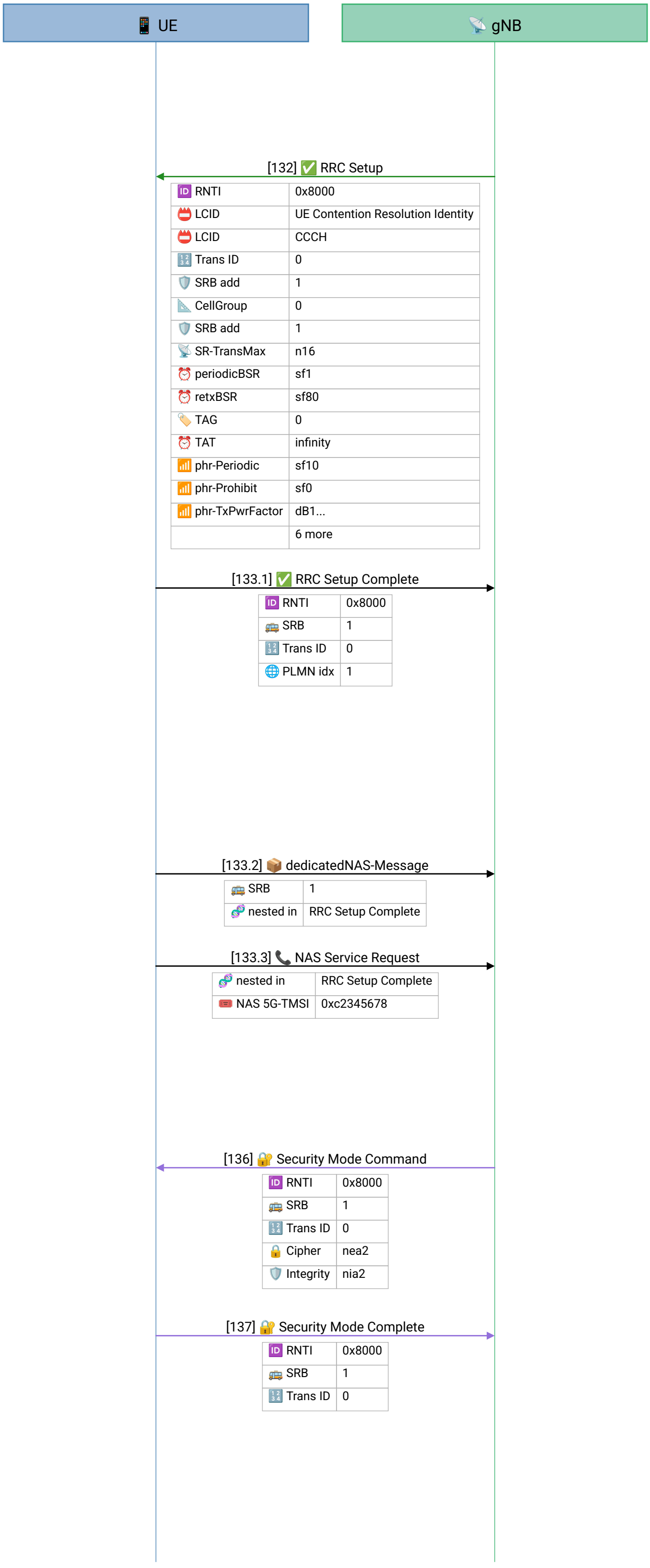
[Frame 129] The same release again — but RRC only sent it once. Identical content, 44.894 ms later. This is not a second `RRCRelease`; it is RLC retransmitting the PDU because its poll went unanswered, and the timeline shows it twice because the frame really is on the wire twice.

Worth keeping straight when counting messages off a diagram: two arrows, one RRC message. The RLC companion explains why the timer fired and what the 44.894 ms measures.

[Frame 131] A second connection — and the identity choice flips. The same C-RNTI `0x8000` is reused, which is why this shares a timeline with everything above, but the RRC connection is brand new: idle since the release, no keys, no bearers.

Compare it field by field with frame 2. Same message type, same 99 bytes on the wire, same 39-bit identity field — and both of the values in it are different in kind:

- `establishmentCause: mt-Access (2)` instead of `mo-Signalling`. The network is reaching for the UE, not the other way round.
- `ue-Identity` takes the *other* branch of the CHOICE: `ng-5G-S-TMSI-Part1 (0)`, `838468acf0`, where



frame 2 took `randomValue`. TS 38.331 makes that choice depend on whether the upper layers hand RRC a 5G-S-TMSI for this particular connection. They did not for an initial registration; they did for a mobile-terminated service request. Frame 133 shows what happens to the rest of that identity.

[Frame 132] The same RRC Setup, byte for byte. 0.711 ms after Msg3 — against 0.708 ms on the first connection. The `masterCellGroup` is 209 octets again and is *identical* to frame 3's, hash for hash: same SRB 1 bearer config, same BSR and PHR timers, same `p-NR-FR1: 23 dBm`, same `t310` / `t311` of `ms1000`. A fresh RRC connection to the same cell gets the same starting configuration. Nothing the first connection learned — not the capabilities, not the bearer setup — carries over into the setup message.

[Frame 133.1] The second half of the identity. 28.257 ms after Msg4, against 28.259 ms on the first connection. `selectedPLMN-Identity: 1`, and then the field that makes this connection's opening make sense: `ng-5G-S-TMSI-Value: ng-5G-S-TMSI-Part2 (1)`, 9 bits, `0000`. A 5G-S-TMSI is 48 bits. Msg3 is sent on CCCH with a fixed, tiny grant, and TS 38.331 gives its `ue-Identity` field 39 bits — not enough. So RRC splits the identity across two messages: 39 bits in Msg3, the remaining 9 bits here, in the first message with a dedicated bearer under it.

And notice what is *gone* compared with frame 4: no `registeredAMF`, no `guami-Type`. It is not needed. The 5G-S-TMSI the UE just finished spelling out contains the AMF Set ID and AMF Pointer, so the gNB can already route the NAS message.

[Frame 133.2] A much smaller container this time. `dedicatedNAS-Message`, **40** octets against frame 4's 92. Same courier role, less to carry: a returning UE resuming an existing context has far less to say than one starting a registration.

[Frame 133.3] And the NAS message inside it. A Service Request, `Security header type: Integrity protected (1)`, message authentication code `0xb98305d2`, NAS sequence number 4. `Service type: Mobile terminated services (2)` — matching frame 131's `mt-Access` establishment cause exactly, one layer up.

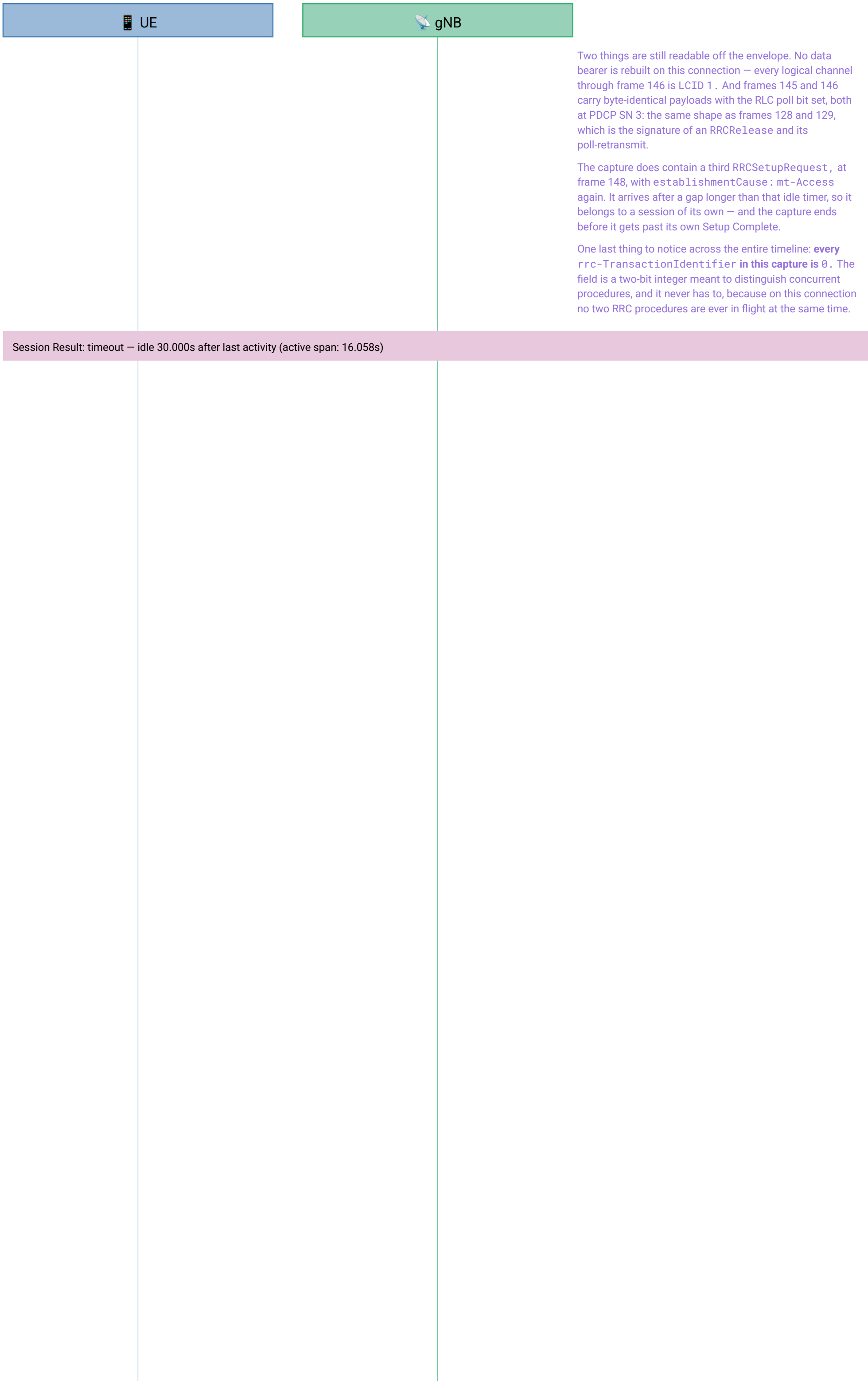
The mobile identity is `5G-S-TMSI` with `5G-TMSI: 0xc2345678` — the same value frame 4's 5G-GUTI carried. The UE never lost its identity across the release; what changed is that this time NAS handed it down to RRC.

[Frame 136] AS security, from scratch, again. `cipheringAlgorithm: nea2`, `integrityProtAlgorithm: nia2` — the same algorithms frame 17 chose, but this is a genuinely new activation. The release at frame 128 destroyed the previous AS context, so a new `K_gNB` is derived and the whole radio-security procedure runs from the beginning.

Every RRC connection secures itself independently. There is no such thing as inheriting radio keys across an idle period.

[Frame 137] Security Mode Complete — and the timeline stops here. 13.228 ms after the command, and empty apart from `rrc-TransactionIdentifier: 0`, exactly as at frame 18.

What follows is the point — and this diagram cannot show it. The session ends here because the extractor's 30-second idle timer closed it, and because the readable RRC ends here too. Frames 139, 140, 142, 145 and 146 are five more `LCID 1` PDCP data PDUs on this same connection, and every one is opaque: the security activation two frames back derived a new `K_gNB`, and the published keys cover the first connection only.



Two things are still readable off the envelope. No data bearer is rebuilt on this connection — every logical channel through frame 146 is `LCID 1`. And frames 145 and 146 carry byte-identical payloads with the RLC poll bit set, both at PDCP SN 3: the same shape as frames 128 and 129, which is the signature of an `RRCRelease` and its poll-retransmit.

The capture does contain a third `RRCSetupRequest`, at frame 148, with `establishmentCause: mt-Access` again. It arrives after a gap longer than that idle timer, so it belongs to a session of its own — and the capture ends before it gets past its own Setup Complete.

One last thing to notice across the entire timeline: **every `rrc-TransactionIdentifier` in this capture is `0`.** The field is a two-bit integer meant to distinguish concurrent procedures, and it never has to, because on this connection no two RRC procedures are ever in flight at the same time.

Session Result: timeout — idle 30.000s after last activity (active span: 16.058s)