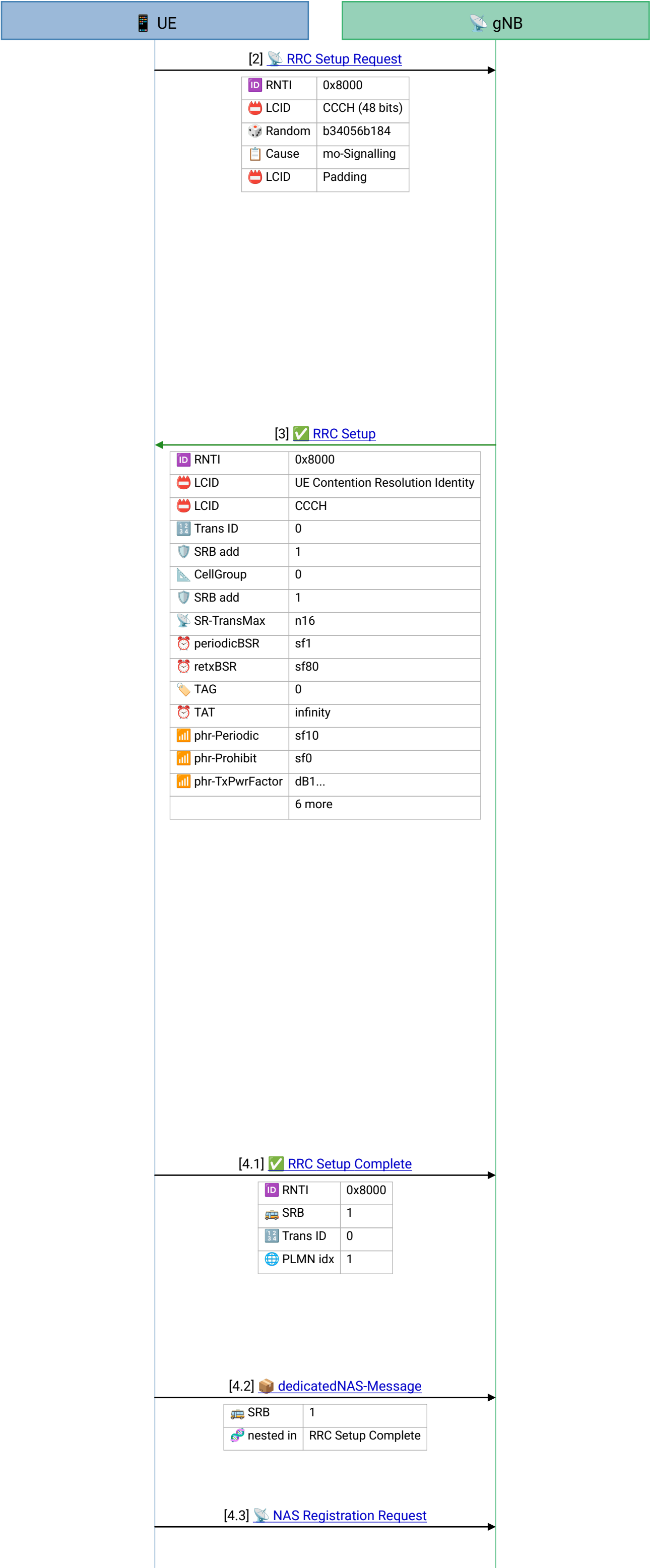
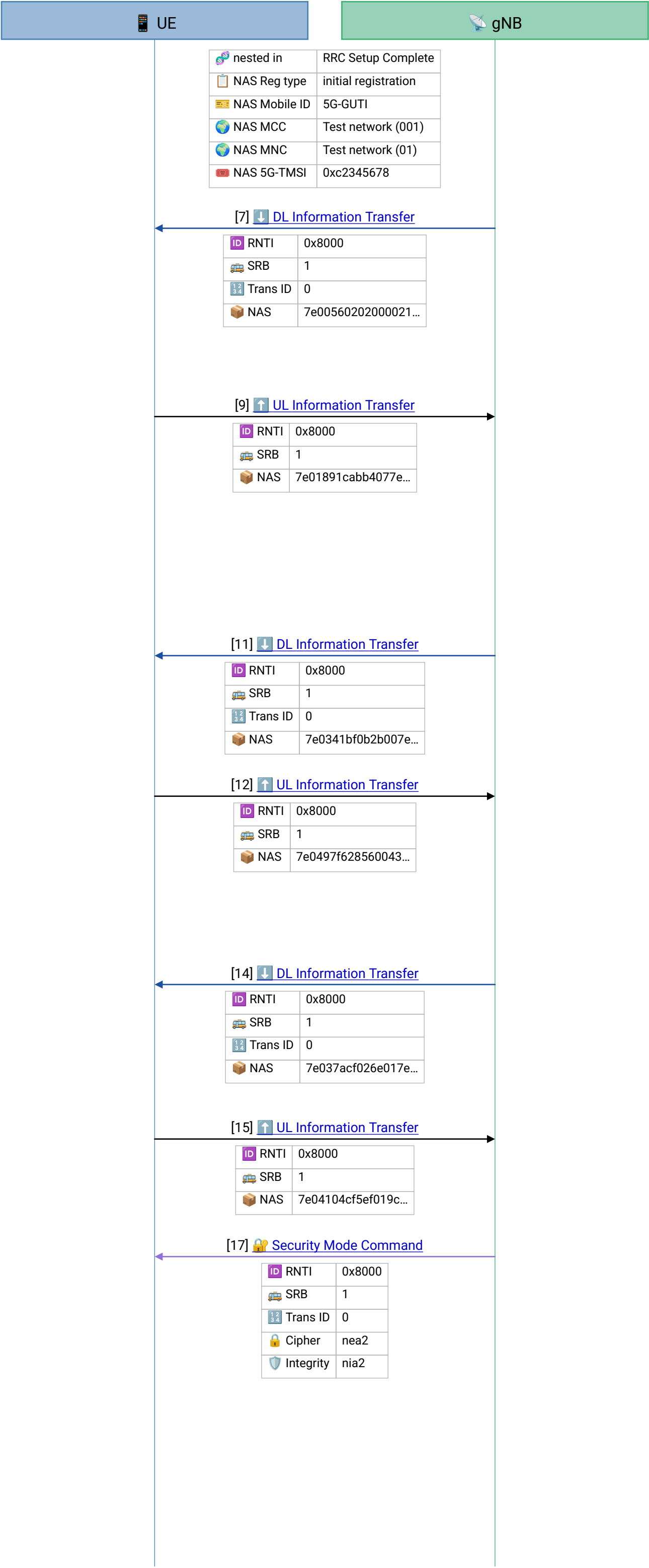






header type: Integrity protected (1), message authentication code 0x97a0a3f4, NAS sequence number 6, and then a plain 5GMM body. 5GS registration type: initial registration (1), mobile identity 5G-GUTI with 5G-TMSI: 0xc2345678. The UE therefore already holds a NAS security context from an earlier visit – 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.

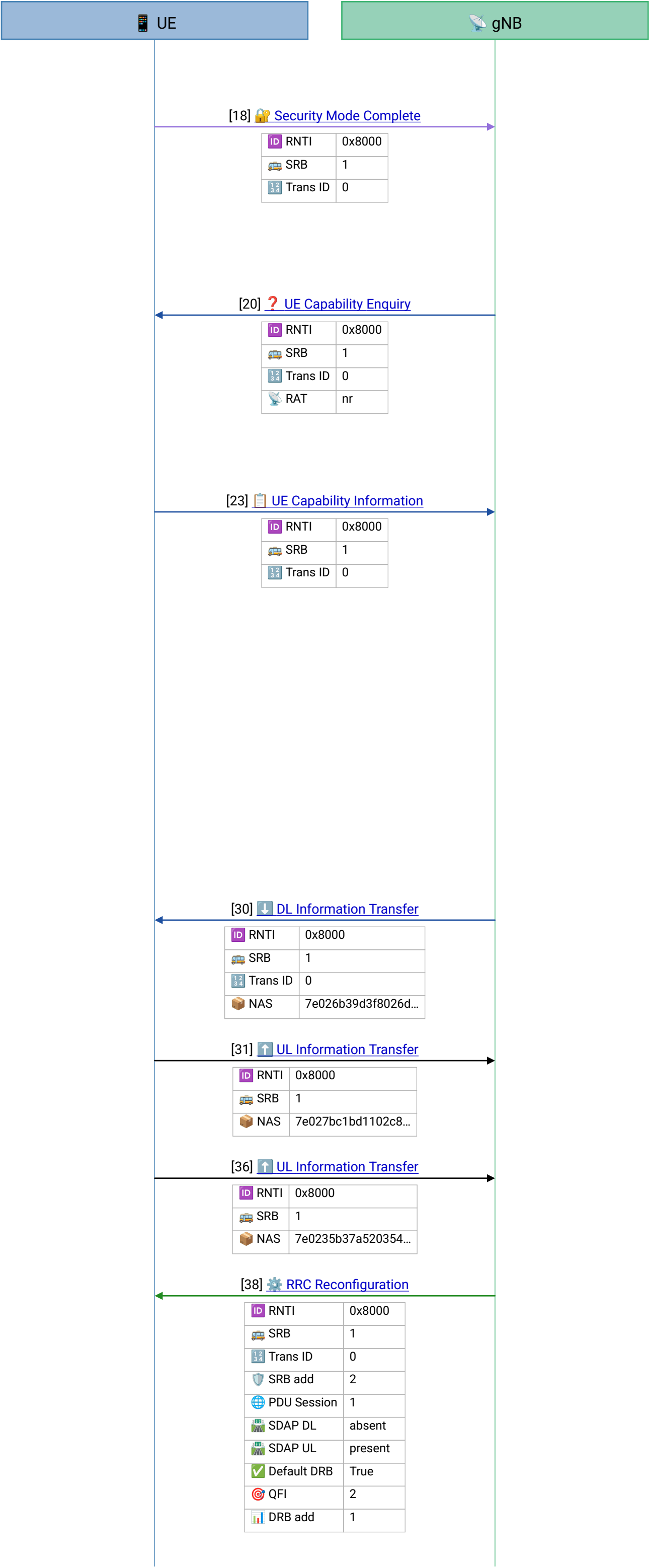
RRC's transaction identifier stays at 0 throughout, because none of this is an RRC procedure at all.

[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.

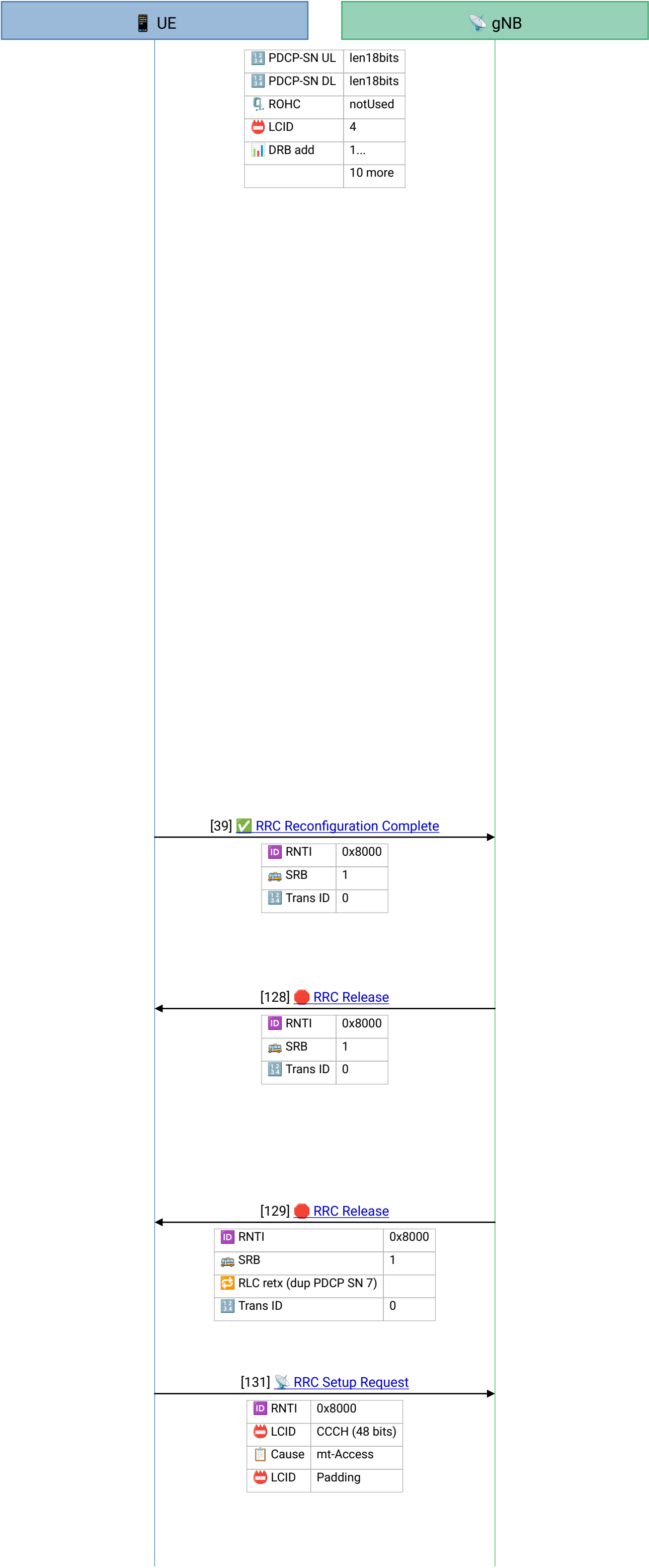
[Frame 31] A ten-octet answer. Ciphered, `security header type 2`. RRC does not know or care whether this is an accept, a reject, or a completion – it is ten octets of somebody else's conversation.

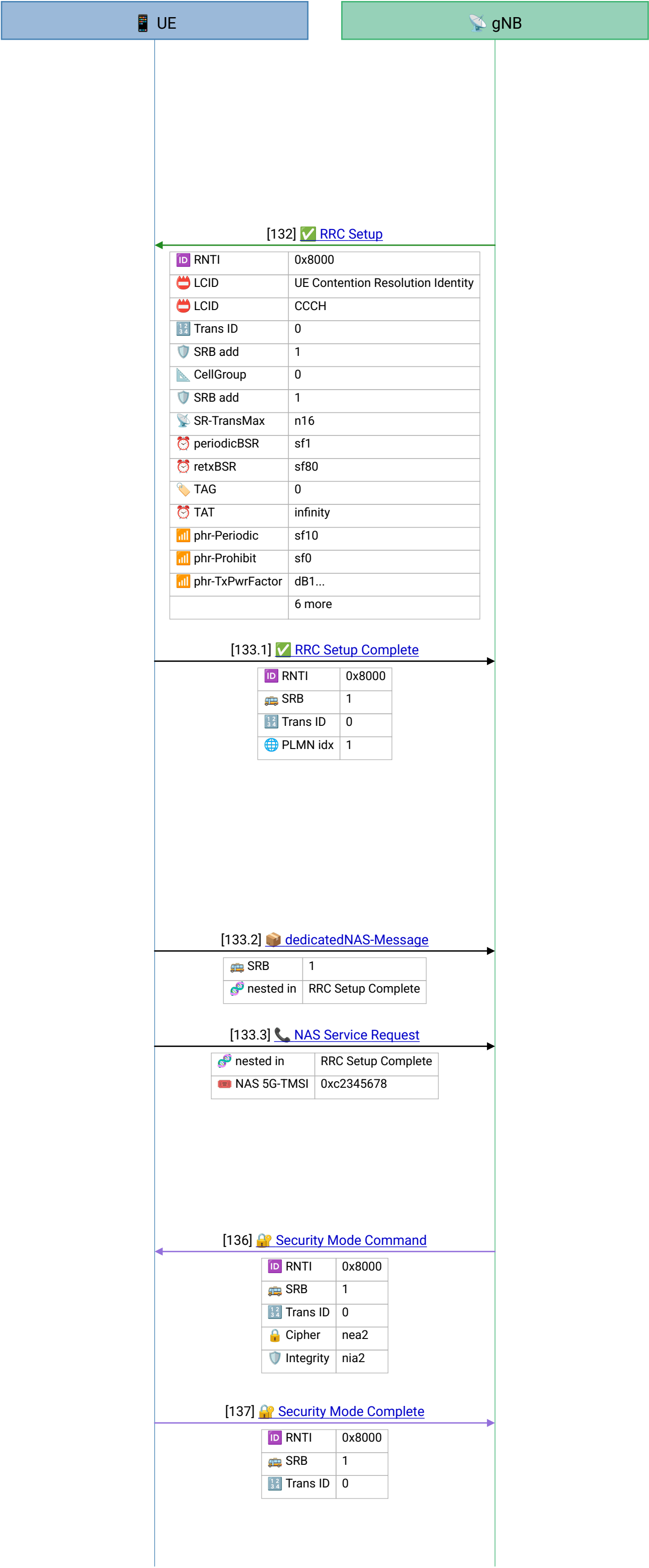
[Frame 36] A 92-octet container, ciphered – the same size as the Registration Request this connection opened with. Whatever session-establishment request it carries, the network's answer arrives two frames later – and that answer is the one RRC message on this timeline that builds a data bearer.

[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





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

 UE

 gNB

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 `RRCRRelease` and its poll-retransmit.

The capture does contain a third `RRCSSetupRequest`, 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)