# Security analysis of DCCF collection profile lifecycle

This report in another language: 中文.

19.7.0 · ProVerif+ Agent report

| | |
|---|---|
| **Documents** | 5G System; Data Collection Coordination Services; Stage 3 (protocol definition) |
| **Properties examined** | 10: 3 hold, 6 fail (7 attacks), 1 without an answer |
| **Findings** | 1 |

**Contents**


1 Protocol and scope
2 Findings
3 Properties examined
4 Limits of this analysis
Appendix A: Reading of each property
Appendix C: Claims and prover answers
Appendix D: How the analysis was run


## 1 Protocol and scope

The Ndccf_ContextManagement service lets an NF service consumer keep its data collection interest at the Data Collection Coordination Function: an NWDAF or an ADRF registers a data collection profile, updates it and deregisters it, so that later data collection can be coordinated from that profile. The profile carries the collection information in anaSub or dataSub together with one consumer identifier, the DCCF assigns the profileId, and the consumer, the DCCF and the NRF authenticate each other over HTTP/2 on the service based interface and on the NRF link, the NRF issuing the access tokens the API relies on.

When the exchange ends the consumer is sure that the DCCF accepted the request it sent, and the DCCF is sure that the request arrived over an authenticated connection from an NF holding a valid token for the ndccf-contextmanagement service. The DCCF is not assured that the consumer identifier inside the request body is that of the NF that sent it, and nothing in the sub-procedure binds a profileId to the consumer that created it.

The parties:

- **NWDAF**: NF service consumer, registers, updates and deregisters its profile. Holds: nwdafId, client credentials, access token.
- **DCCF**: provides Ndccf_ContextManagement, keeps the profiles. Holds: profiles (profileId, consumer identifier, collection information), authorization server public key.
- **NRF**: authorization server, issues access tokens. Holds: client credentials of the NFs, signing key.

Figure 1 shows the exchange as the documents specify it.

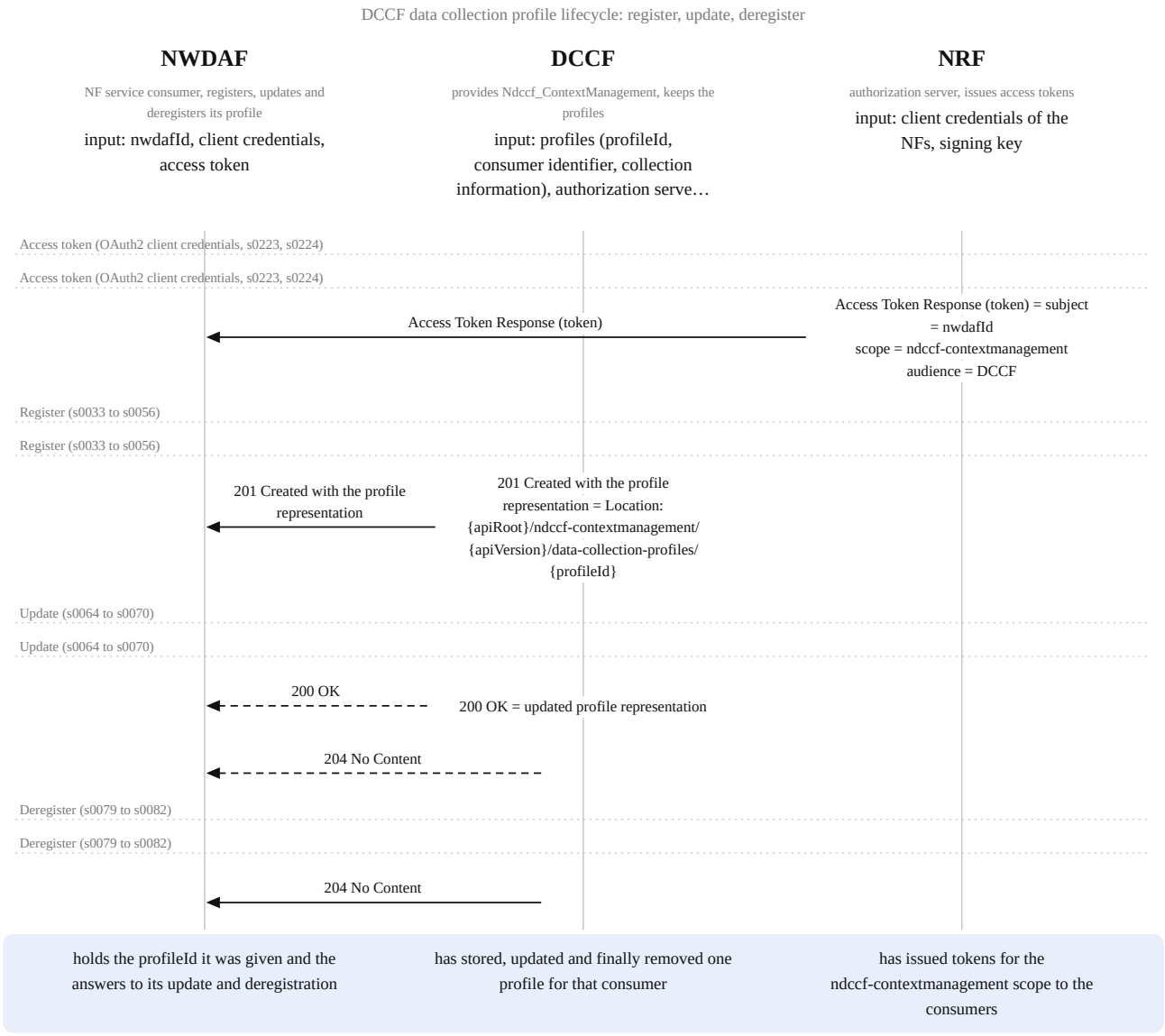



Figure 1. The protocol as specified.

The scope of the analysis is in Appendix D: what was analysed, what was left out, and the sections of the documents it is built from.

## 2 Findings

The analysis has 1 finding: what follows is not stated in the documents. For 1 of them it is unclear whether the documents state it.

The token checks hold: every registration, update and deregistration the DCCF commits is backed by a token the NRF issued to that NF for the ndccf-contextmanagement scope, and the collection information reaches only the DCCF. What fails is the binding between a request and the profile it names: the DCCF stores whatever consumer identifier the body carries without comparing it with the authenticated consumer, and it accepts an update or a deregistration of another NF's profile whenever the profileId is known or predictable. That is a finding the documents do not state, because they record one service-wide scope and no ownership rule, so the profileId silently

carries the authorization. Of the ten properties examined, all have an answer under both attackers, six are answered only in part with the failing half shown as an attack, and two rest on an unrolled lifecycle that ProVerif could not fully certify; Tamarin+ Agent would settle those state questions.

The attack figures use the columns of Figure 1. Red is what the attacker does. Yellow marks a message that a party sends although the specified exchange does not have it at that point, and the outcome that the attack breaks.

## 2.1 Any NF holding a service-scope token can write into another NF's profile

| | |
|---|---|
| **Kind** | Weakness of the protocol as specified |
| **New** | Unclear: the documents state it in part, or could not be searched |
| **Value** | medium |
| **Properties** | A02, A03, A08, A08.7, Z03, Z03.8 (section 3) |
| **Source text** | s0033, s0047, s0052, s0064, s0226 |
| **Shown by** | ProVerif attack queries `A03_bind`, `A08_owner`, `A08_owner_b`; Figures 2, 3, 4, 5 and 6 |

**What happens.** The DCCF checks the access token's signature, scope and audience, then stores or replaces a profile without comparing the consumer identifier in the request body with the subject of the token. A second legitimate NF can therefore register a profile that names another NF, for example the NWDAF's nwdafId. With the profileId known or predicted it can also update or deregister a profile recorded under another consumer's identifier, and the DCCF accepts.

**What the attacker needs.** Only the client credentials of a genuine NF service consumer instance and the access token the NRF issues for the service scope. To change another NF's profile it must also know or predict the profileId, which the source leaves entirely open: it says the DCCF assigns one and nothing about how.

**What the attacker gains.** It can make another consumer's registered collection request say what it wants, or delete another consumer's registration, and it can put any consumer identifier into a profile so that the stored profile misstates who registered the collection.

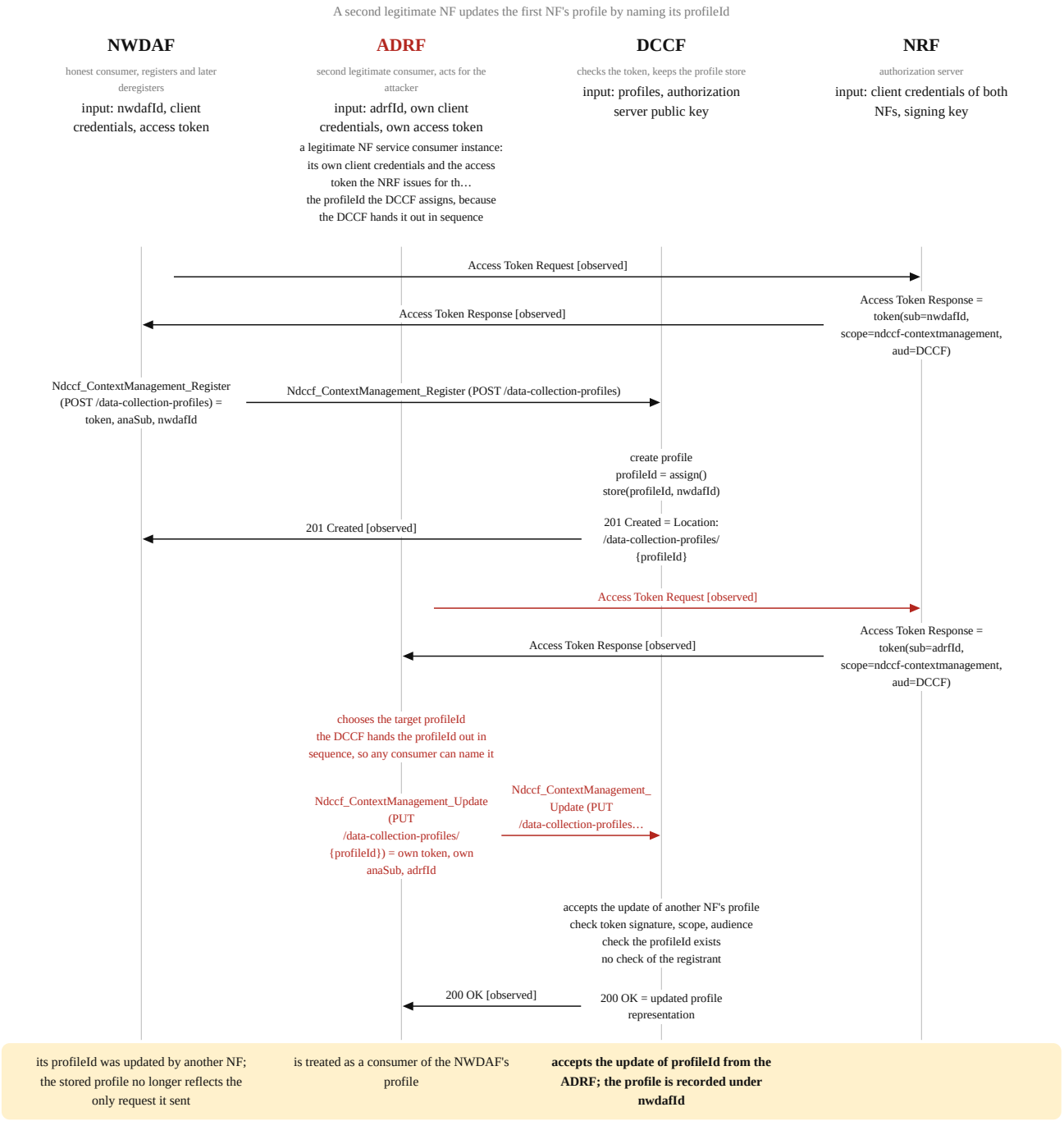



Figure 2. A second legitimate NF updates the first NF's profile by naming its profileId. Property Z03.8; attack queries A08_owner, A08_owner_b.

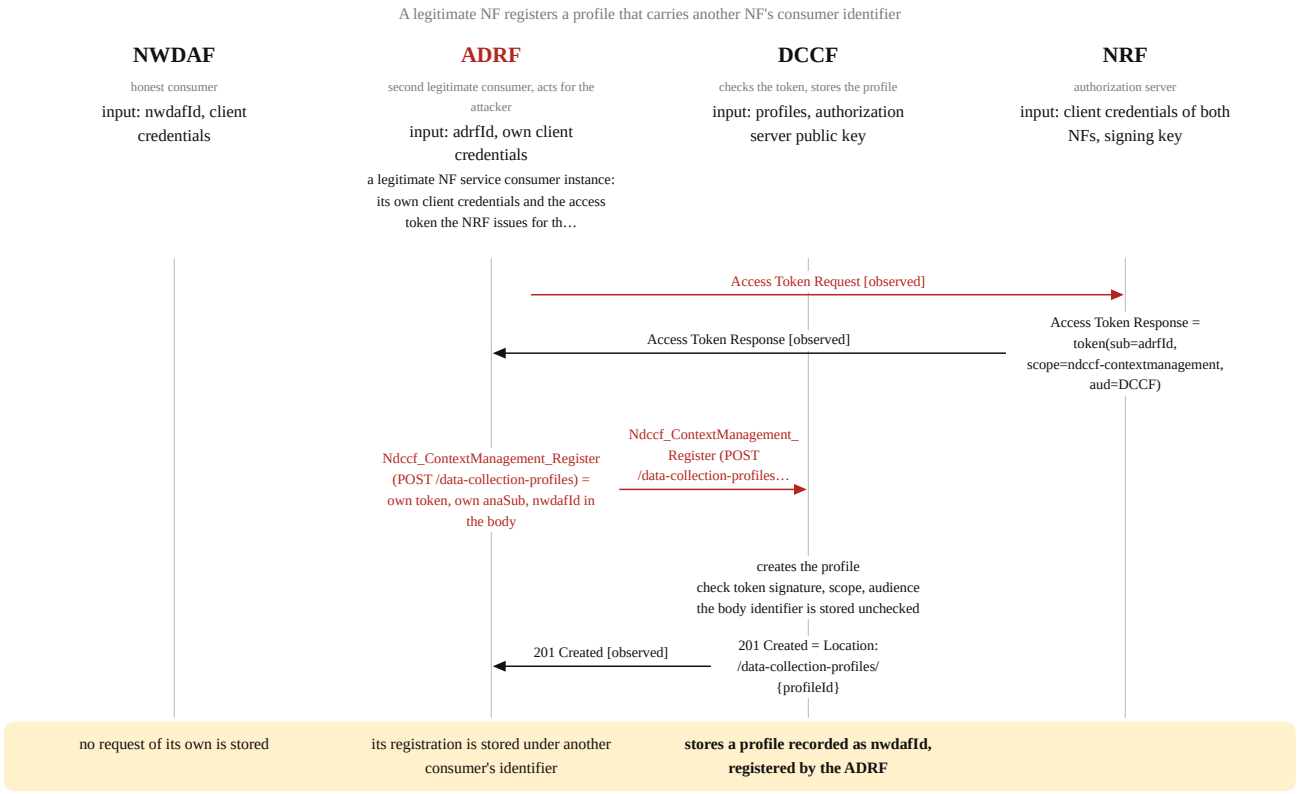



Figure 3. A legitimate NF registers a profile that carries another NF's consumer identifier. Property A03; attack query A03_bind.

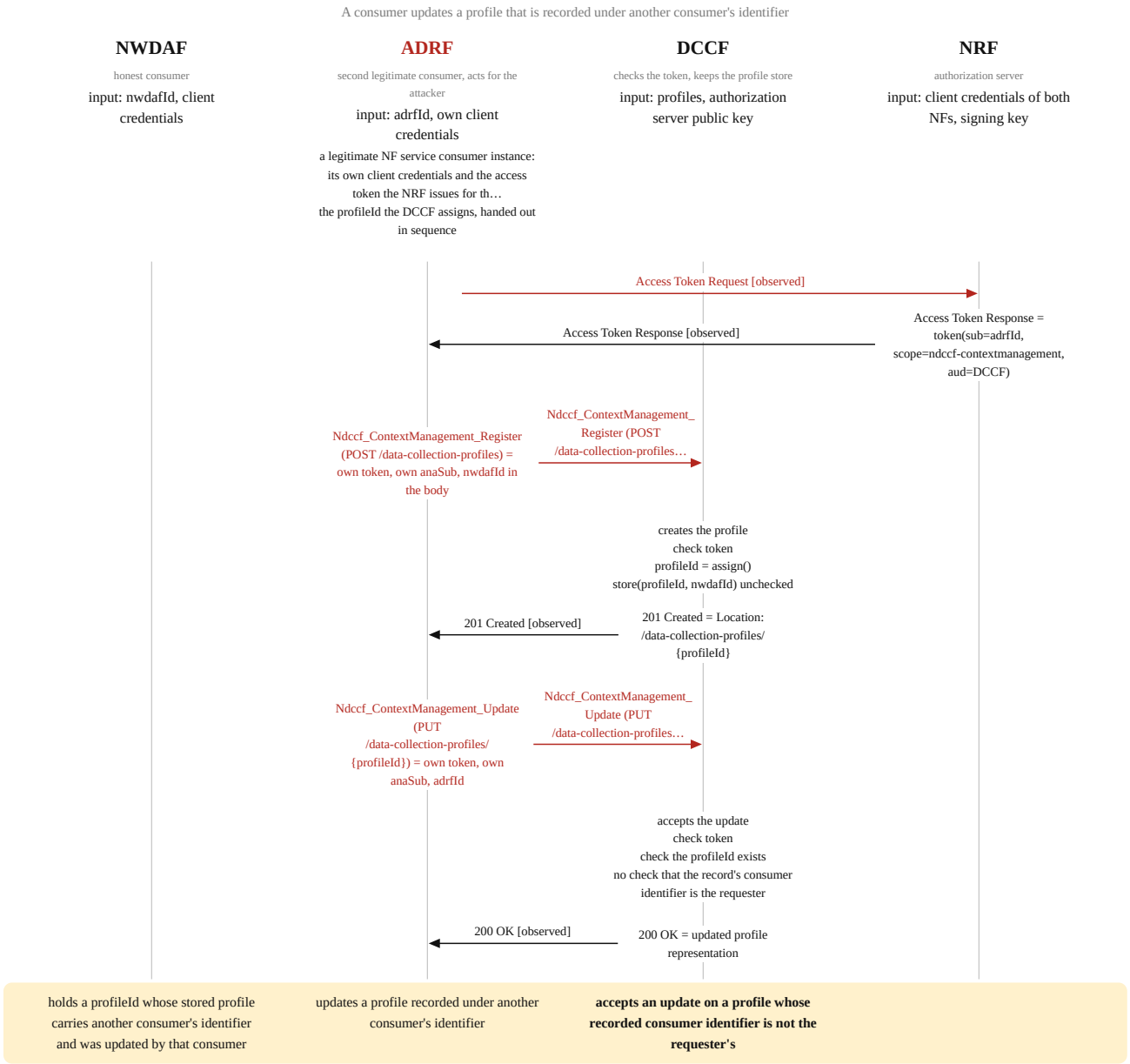



Figure 4. A consumer updates a profile that is recorded under another consumer's identifier. Property A08; attack query A08_owner.

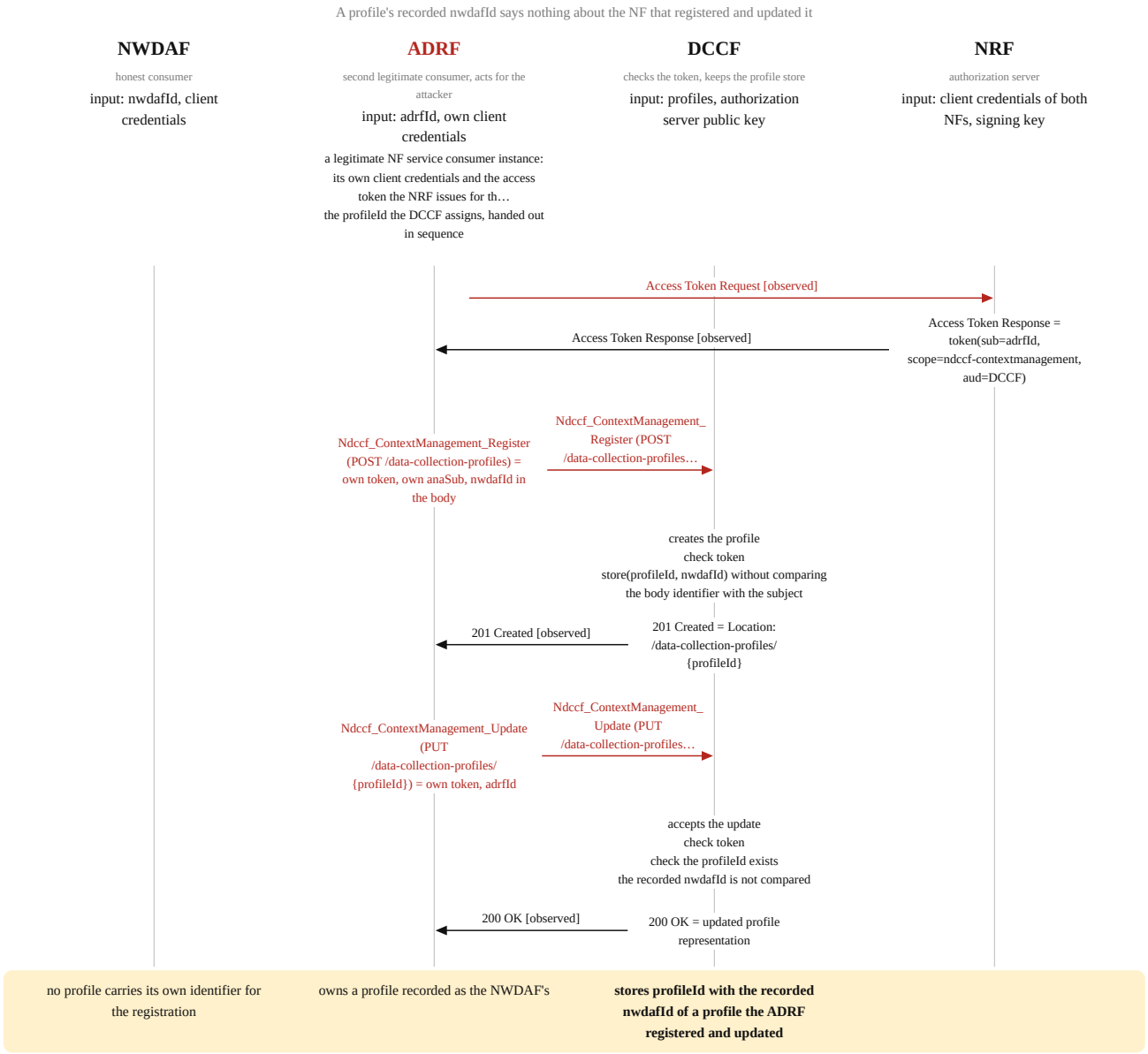



Figure 5. A profile's recorded nwdafId says nothing about the NF that registered and updated it. Property A08.7; attack query A08_owner.

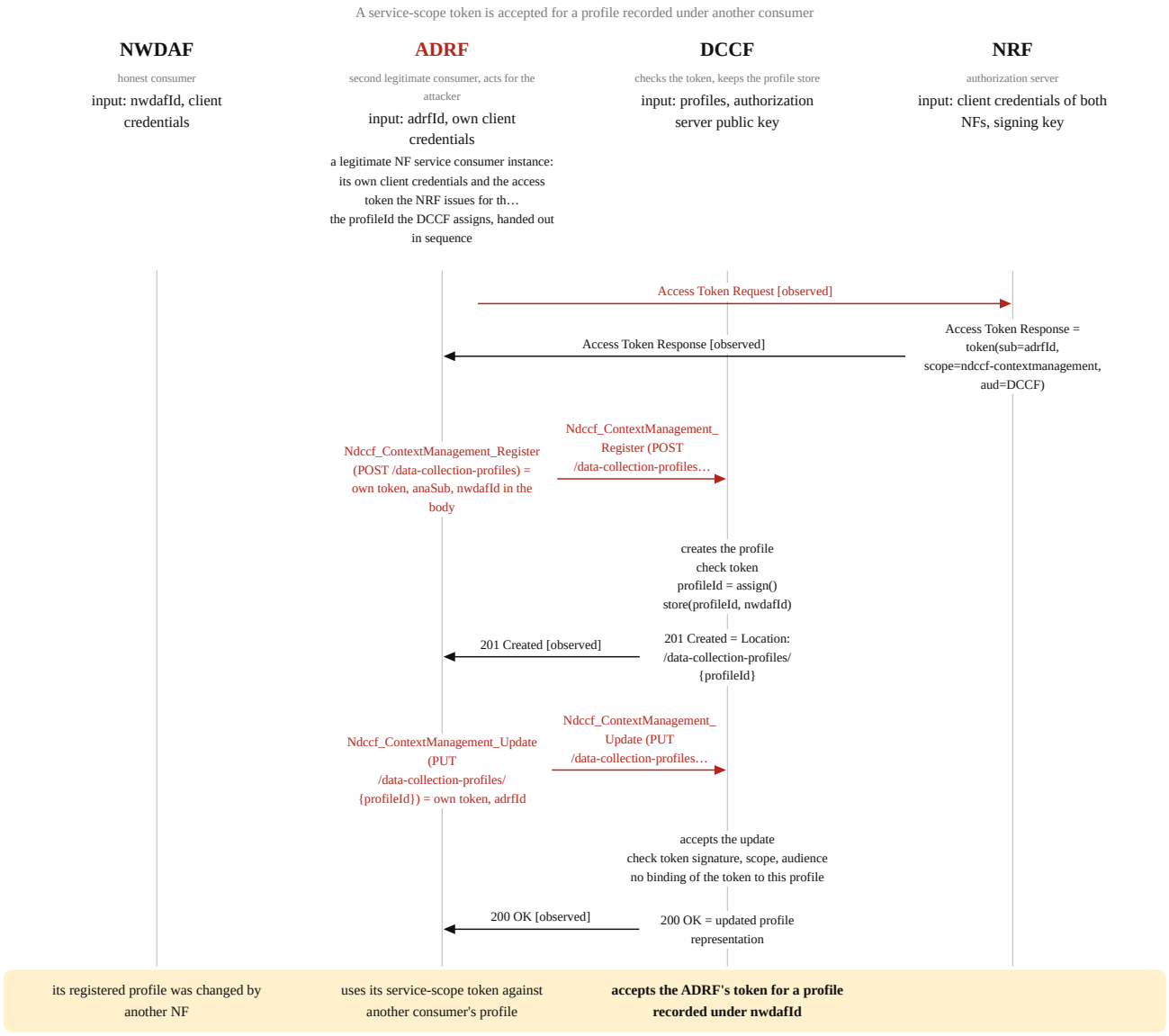



Figure 6. A service-scope token is accepted for a profile recorded under another consumer. Property Z03; attack query A08_owner.

**Proposed change.** Make the DCCF compare the consumer identifier in the NdccfDataCollectionProfile body (nwdafId, adrfId, nwdafSetId or adrfSetId) with the subject of the presented access token and reject the request on a mismatch. Bind every created profile to that subject, and reject a PUT or DELETE whose token subject is not the subject recorded at registration with 403 before the profile is touched.

**Check of the change with ProVerif.** Not checked: The change is a new check inside the DCCF, not a change of the messages, and the source leaves the refusal of an invalid request to clause 5.2.7; a model with the check would have to invent that refusal and would no longer be a model of this text. No model of the specified text can remove the weakness.

No figure was given for these attacks of a finding: A02: A03_bind, A02: A08_owner, A02: A08_owner_b, A03: A08_owner, A03: A08_owner_b, A08.7: A03_bind, A08.7: A08_owner_b, A08: A03_bind, A08: A08_owner_b, Z03.8: A03_bind, Z03: A03_bind, Z03: A08_owner_b.

# 3 Properties examined

Each row is one property put to ProVerif. The result is ProVerif's answer on the analyst's model of the protocol; value and whether the documents already state the result are the analyst's judgement, with the reasons in Appendix A.

| ID | Property | Compromise assumed | Result | Value | In the documents |
|---|---|---|---|---|---|
| A02 | **Authenticated initiator of each request.** Whenever the DCCF accepts a registration, update or deregistration request, the claimed initiator (a NWDAF or ADRF instance) must actually have run this service under that identity. | no compromise | Fails (1 attack) | medium | unclear |
| A03 | **Registered consumer identifier matches the sender.** Whenever the DCCF accepts a registration, a matching consumer run exists, and the two agree on the collection information (anaSub/dataSub) and on the consumer identifier (nwdafId/adrfId/nwdafSetId/adrfSetId). | no compromise | Fails (1 attack) | medium | unclear |
| A08 | **profileId bound to its creating consumer.** Every profileId the DCCF accepts corresponds to the one profile created by that consumer at registration, and its collection information and consumer identifier are the same as at creation. | no compromise | Fails (1 attack) | medium | unclear |
| Z01 | **Accepted operations carry a valid service token.** Every registration, update or deregistration operation the DCCF commits must carry a token issued by the authorization server that is valid for the ndccf-contextmanagement service. | no compromise | Holds | low | unclear |
| Z03 | **Token bound to subject, resource and operation.** A token may be used only for the consumer identity declared by its authorized subject, for the profileId it belongs to, and for the authorized operation type; it cannot be used for another NF's profile or for another operation. | no compromise | Fails (1 attack) | medium | unclear |
| S05 | **Deregistered profileId no longer valid.** After the point at which deregistration takes effect, a deleted profileId can no longer trigger an update of that profile or be treated as a valid registration. | no compromise | Reachable only | low | unclear |
| A08.7 | **profileId bound to creator and to nwdafId.** Every profileId the DCCF accepts corresponds to the one profile created by that consumer at registration, and its nwdafId/adrfId is the same as at creation. | no compromise | Fails (1 attack) | medium | unclear |
| Z03.8 | **Update and deregister authority over a profile.** A token may be used only for the consumer identity declared by its authorized subject, for the profileId it belongs to, and for the authorized operation type; it cannot be used for another NF's profile or for another operation. | no compromise | Fails (2 attacks) | high | unclear |
| U01 | **Responses match requests and deletion is final.** The registration/update responses match the corresponding requests; after deregistration the old profileId is no longer valid. | no compromise | Holds | low | unclear |
| X01 | **Consumer authenticates the DCCF endpoint.** No NF service consumer ever sends its access token or its data collection profile, including the collection information in anaSub/dataSub, to a party other than the DCCF instance it is addressing: the endpoint it talks to is authenticated as the DCCF, so a party that is not the DCCF never receives a request of this service. | a party on the network that answers on the DCCF's address | Holds | low | yes |

Goals the documents state that the drafted list of properties did not have, added by the analyst (Appendix D): X01.

The questions asked, and the analyst's answers:

| Question | Answer | Rests on |
|---|---|---|
| Q1: Binding of profileId to creator identity and to nwdafId/adrfId | No. The DCCF records whatever consumer identifier the request body carries and never compares it with the authenticated consumer, so a profile registered by the ADRF can say nwdafId = NWDAF; the probe is target A08.7, whose attack holds both in the multi-session model and on the unrolled lifecycle. | A08.7, A08, A03 |
| Q2: Whether another legitimate NF's update/delete authority comes from an explicit rule in the specification | No. The only authorization rule in the source is the single service-wide scope ndccf-contextmanagement, and neither the update nor the deregister operation carries a check of the registrant, so another NF's profile is reachable by any NF that knows the profileId; that is the weakness reported for Z03.8, Z03 and A08. | Z03.8, Z03, A08 |
| Q3: Registration/update response correlation and the bounded-state behaviour after deregistration | The responses are correlated with the requests: the consumer completes a lifecycle only when the profileId it received in the 201 comes back in the update and deregistration responses, and that holds. For the state after deregistration the source only says the profile is removed; on the unrolled lifecycle the store no longer holds the profileId and a later request for it is refused, but ProVerif could not certify that the refusal is reachable, so the deletion question stays partly open (targets U01 and S05). | U01, S05 |

# 4 Limits of this analysis

A result here is a result on a model of the protocol: it says what the specified messages and checks give or fail to give, with cryptography taken as ideal, and nothing about an implementation.

What this analysis does not cover, as the analyst states it:

- TLS on the service based interface and on the NRF link is modelled as one authenticated confidential channel per endpoint pair, so the network attacker can neither read, modify nor inject that traffic; only a party that is a genuine NF threatens the messages.
- The access token is a signature over subject, scope and audience; expiry, caching and revocation are not modelled, and the model's authorization server issues only whole-service-scope tokens, so scope confusion between services appears only as a token for another service.
- In model.pv the DCCF's profile store is a ProVerif table, which only grows, so the removal a deregistration performs cannot be represented and the profileId is a fresh unguessable name; the lifecycle is therefore also modelled in bounded.pv, unrolled to four steps, with threaded state and a profileId handed out in sequence.
- In bounded.pv ProVerif could not reconstruct a trace for the last steps of the chain, so the supply of the unrolled model's later steps is not certified; Tamarin+ Agent would settle those questions.
- The request body is reduced to a consumer identifier and one collection-information value; the structured NdccfDataCollectionProfile type, its mandatory attributes and the error responses of clause 5.2.7 are not modelled.
- The analysis covers this sub-procedure only: collection of data or analytics from the sources, notifications, retrieval, and transfer between DCCFs are outside the scope.
- Attackers beyond the network and a legitimate NF with its own credentials are not modelled: a stolen token presented from another endpoint, an NRF that mints tokens, and a dishonest DCCF are not answered by any query.
- Only properties of traces are established: nothing is said about availability, throughput, or the computational strength of the signature, the token or TLS.

What the host records:

- No answer for: S05.
- The answers of A08, Z03, S05, A08.7, Z03.8, U01 rest on an operation the model simplifies (Appendix D).

- Set by the analyst and without a proof or an attack (Appendix D): S05 under `network`; S05 under `legit_nf`.
- Proved without a control query (Appendix C): A02: A02_auth_upd, A02: A02_auth, A08: A08_created_b, Z01: Z01_upd, Z01: Z01_reg, U01: A03_body, X01: X01_secret. A control is a variant of the property that must fail; without one, nothing shows that the property as written can fail in this model.

What stays open for each property, and what would settle it, is in Appendix A.

# Appendix A: Reading of each property

The analyst's own reading of the prover's answers, not an answer of the prover: what the answers mean for the protocol, what the reading rests on, whether it can be used as it stands, and what it is worth.

## A02: Authenticated initiator of each request

Whenever the DCCF accepts a registration, update or deregistration request, the claimed initiator (a NWDAF or ADRF instance) must actually have run this service under that identity.

Every accepted registration, update and deregistration is backed by a token the NRF issued to the NF that presents it, so no outsider and no NF can act under an identity it does not hold. What is not assured is the identity written inside the profile: an NF can register a profile that says another NF's nwdafId or adrfId, because the DCCF never compares the two.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value medium: The proof covers the half of the target the source requires; the attack on the other half is a shift of the consumer identity that the source does not forbid in so many words.
- Stated in the documents: unclear. Source text it rests on: ts-33-501:2871, s0026, s0033, s0064, s0079
- Rests on: TLS is modelled as an authenticated confidential channel per endpoint pair.; The token is a signature over subject, scope and audience; expiry is not modelled.
- Open: Nothing in the source says whether the DCCF compares the body identifier with the authenticated identity; the claim that it does is attacked, not proved.
- Next step: Ask the same claim of the deregistration in a model where an SCP sits between the parties, as the note to TS 33.501 clause 5.9.2.1 allows.

## A03: Registered consumer identifier matches the sender

Whenever the DCCF accepts a registration, a matching consumer run exists, and the two agree on the collection information (anaSub/dataSub) and on the consumer identifier (nwdafId/adrfId/nwdafSetId/adrfSetId).

The collection information stored is exactly what the registering NF sent, so nobody can supply or alter it along the way. The consumer identifier stored next to it is not tied to the sender: the ADRF can register a profile that reads as the NWDAF's request.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value medium: A reader of the stored profile is misled about who registered the collection; the source's shall clause covers the body's content but not who wrote it.
- Stated in the documents: unclear. Source text it rests on: s0033, s0047, s0052, s0056, s0196
- Rests on: TLS is an endpoint channel; the SBI transport is not attacked.; The body is modelled as a consumer identifier plus one collection-information value.
- Open: Whether the DCCF is required to compare the body identifier with the authenticated consumer.
- Next step: Have the editors decide whether the body identifier must equal the token subject, then either add the check or state the freedom.

### A08: profileId bound to its creating consumer

Every profileId the DCCF accepts corresponds to the one profile created by that consumer at registration, with the same collection information and consumer identifier as at creation.

On the multi-session model a profile is only updated when the DCCF already holds it, so an update cannot invent a profile; the profileId is fresh, and an NF cannot learn another NF's value, so cross-NF writing is not reachable there. On the unrolled lifecycle, where the profileId is handed out in sequence, the ADRF updates the NWDAF's profileId and the DCCF accepts it.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value medium: It isolates the gap: the source binds neither the profileId to its creator nor the request to the registrant, and the answer turns on an assumption the source leaves open.
- Stated in the documents: unclear. Source text it rests on: s0054, s0056, s0064, s0148
- Rests on: On model.pv the profileId is a fresh unguessable name, which the source does not require; on bounded.pv it is predictable, which the source does not forbid.; On bounded.pv the chain has four steps and ProVerif could not certify that its later steps are reachable.; The update replaces the stored contents, but the model does not carry them through the update.
- Open: Whether a profileId can be guessed, which decides whether another NF can address a profile at all; the source says nothing about its generation.
- Next step: Model the stored contents across an update, so that the overwrite of another NF's profile is visible as a change of what the DCCF will act on.

### S05: Deregistered profileId no longer valid

After the deregistration takes effect, the deleted profileId can no longer trigger an update of that profile or be treated as a valid registration.

On the unrolled lifecycle the deregistration empties the DCCF's store, so a later request naming that profileId finds no profile and is refused instead of re-registering it. The multi-session model can only show that a deregistration is accepted at all, because its table cannot forget a profile; the falsification it reports is an artifact of that table.

- Result: Reachable only; answers what is asked: partly; usable with caveats
- Value low: The answer concerns a state ProVerif can express only in an unrolled chain, and the conclusive part could not be certified.
- Stated in the documents: unclear. Source text it rests on: s0079, s0080, s0081, s0082
- Rests on: Only four steps of the lifecycle are unrolled, and ProVerif could not reconstruct a trace for the last of them.; The source does not say whether a profileId may be reused after deletion.
- Open: Whether a deregistered profileId is refused or silently re-created when a later request names it; the unrolled model's refusal could not be certified as reachable.
- Next step: Rebuild the unrolled chain so that ProVerif can reach the deregistration and the later update, for instance by giving each request its own channel, then ask the post-deletion claim again.

### A08.7: profileId bound to creator and to nwdafId

Every profileId the DCCF accepts corresponds to the one profile created by that consumer at registration, with the same nwdafId/adrfId as at creation.

The profile carries an identifier chosen by the sender, so a reader of the profile cannot rely on it to know who registered the collection. The profileId does not restore that: a consumer that did not register the profile can update it.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value medium: It answers the question a reader of the profile would ask, whether nwdafId says who registered, and the answer is no.
- Stated in the documents: unclear. Source text it rests on: s0047, s0054, s0056, s0064, s0148, s0196
- Rests on: The profile body is modelled as a consumer identifier plus collection information.; ProfileId predictability differs between the two models.
- Open: Whether the changed identifier also changes what the DCCF collects is outside the analysed sub-procedure.
- Next step: Ask the same claims on a model whose store keeps the profile contents, so that the overwrite of the NWDAF's registered collection information is visible.

### Z01: Accepted operations carry a valid service token

Every time the DCCF commits a register, update or deregister operation, it must have a token issued by the authorization server that is valid for the ndccf-contextmanagement service.

No operation is committed without a token the NRF signed for the ndccf-contextmanagement scope and for this DCCF, and a token the same NRF issued for another service is refused. That is the whole of the check: the token is not bound to a profile, so it opens every profile the DCCF holds.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: It confirms what the specification and RFC 6749 require of the authorization step; the result was not in doubt.
- Stated in the documents: unclear. Source text it rests on: rfc6749:2870, ts-33-501:2883, s0223, s0224, s0226
- Rests on: The token is a signature over subject, scope and audience; expiry, caching and revocation are not modelled.; The NRF issues only whole-service-scope tokens in the model.
- Open: Operation-type binding: the API defines one scope for register, update and deregister, so a token is not bound to an operation either; the source states this and no query separates the operations.
- Next step: Add token expiry and a second authorization server, which the client-credentials grant allows, to see whether a token from another NRF is accepted.

### Z03: Token bound to subject, resource and operation

A token may only be used for the consumer identity declared by its authorized subject, the profileId it belongs to, and the authorized operation type; it may not be used for another NF's profile or another operation.

A token is not bound to a profile: the DCCF accepts the ADRF's token for a profile whose recorded consumer identifier is the NWDAF's, and on the unrolled lifecycle for a profileId the NWDAF registered. The single service-wide scope and the absence of any ownership check make a token a key to every profile the DCCF holds, while a party on the network alone gets nothing committed.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value medium: It shows mechanically what the single scope means for this API, which the documents state as a scope decision but not as a consequence for the profiles.
- Stated in the documents: unclear. Source text it rests on: s0064, s0079, s0226
- Rests on: The profileId must be known or predictable for the cross-NF part.; Audience confusion from another DCCF instance is not modelled.

- Open: The cross-NF part needs the profileId known or predictable, which the source neither requires nor excludes.
- Next step: Ask the claim on a model whose store keeps contents, so that the overwrite of another NF's collection information can be read off the trace.

### Z03.8: Update and deregister authority over a profile

A token may only be used for the consumer identity declared by its authorized subject, the profileId it belongs to, and the authorized operation type; it may not be used for another NF's profile or another operation.

Nothing in the sub-procedure stops one NF from modifying or deleting another NF's registration once it knows the profileId; on the unrolled lifecycle the ADRF does exactly that against the NWDAF's profile and the DCCF accepts. Only the profileId's unpredictability stands in the way, and the source does not require it to be unpredictable.

- Result: Fails (2 attacks); answers what is asked: partly; usable with caveats
- Value high: A service that lets any authorized consumer overwrite or delete another consumer's registration has no authorization model for the profiles it defines, and the documents never say the profileId is a secret that carries it.
- Stated in the documents: unclear. Source text it rests on: s0064, s0079, s0226
- Rests on: On bounded.pv the profileId is predictable by construction; on model.pv the attacker cannot obtain another NF's profileId.; The modification is accepted, but the model does not show the resulting contents.
- Open: Whether a deployment relies on the profileId being unguessable; the source is silent.
- Next step: Model the stored contents so that the modification of another NF's registration is visible as a change of what the DCCF will act on.

### U01: Responses match requests and deletion is final

Register/update responses match the corresponding request; after deregistration the old profileId is no longer valid.

The consumer accepts a 201, 200 or 204 as the answer to its own request: it completes its lifecycle only when the profileId it was given comes back in the later responses. Whether the profileId is dead after the deregistration is not settled by the answers recorded here.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: The correlated half follows from how requests and responses are paired; the state half is left open.
- Stated in the documents: unclear. Source text it rests on: s0056, s0068, s0069, s0070, s0080, s0081, s0082
- Rests on: Response correlation is carried by the model's endpoint channels, which stand for the request and response pairs of HTTP/2 over TLS.
- Open: Whether old profileId is refused after the accepted deregistration; S05 covers the same question.
- Next step: Build the unrolled run so that ProVerif can certify the post-deletion refusal, then answer the second half of this target there.

### X01: Consumer authenticates the DCCF endpoint

No NF service consumer ever sends its access token or its data collection profile, including the collection information in anaSub/dataSub, to a party other than the DCCF instance it is addressing: the endpoint it talks to is authenticated as the DCCF, so a party that is not the DCCF never receives a request of this service.

A consumer's token and its collection information reach only the DCCF instance it addresses; no other party of the model, including the other legitimate NF, obtains the collection information the NWDAF registered. This holds because the transport is authenticated between the two endpoints, which is how the interface is specified.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It confirms the transport assumption the whole analysis rests on, and the same goal is lost as soon as an intermediary is inserted.
- Stated in the documents: yes. Where the documents say it: s0099, s0100, s0104
- Rests on: TLS is modelled as an authenticated confidential channel between the two endpoints, which is the assumption the result proves nothing beyond.
- Open: Direct communication only: TS 33.501 clause 5.9.2.1 says mutual authentication is absent when an SCP or SEPP sits between the parties.
- Next step: Ask the same claim in a model where an intermediary such as an SCP shares an endpoint with a consumer.

## Appendix C: Claims and prover answers

The prover's record for each target: the level reached, the query that reached it, whether a control was checked, and the attacks with the analyst's verdict on each.

| target | level | bound | lemma | control | attacks | claims |
|---|---|---|---|---|---|---|
| A02 | proved | | A02_auth_upd | 2 of 4 | A02_claimed (depends_on_ambiguity) | 4 proved, 1 attack |
| A03 | proved | | A03_body | checked | A03_bind (depends_on_ambiguity) | 1 proved, 1 attack |
| A08 | bounded | the changing value unrolled to 4 steps | A08_created_b | 1 of 2 | A08_owner (holds_in_source) | 2 bounded, 1 attack |
| Z01 | proved | | Z01_upd | 2 of 4 | | 4 proved |
| Z03 | proved | | Z01_upd | checked | A08_owner (holds_in_source) | 1 proved, 1 attack |
| S05 | reachable | | run | | S05_del (model_artifact) | 2 reachable, 1 attack |
| A08.7 | bounded | the changing value unrolled to 4 steps | A08_created_b | checked | A08_owner (holds_in_source) | 1 bounded, 1 attack |
| Z03.8 | proved | | Z01_upd | checked | A08_owner (holds_in_source); A08_owner_b (depends_on_ambiguity) | 1 proved, 2 attack |
| U01 | proved | | A03_body | 1 of 2 | | 2 proved |
| X01 | proved | | X01_secret | | | 2 proved |

Claim by claim, in the analyst's wording of each claim; the answer to each is the prover's:

| target | claim | attacker | answer | query | control |
|---|---|---|---|---|---|
| A02 | An accepted update is sent by the NF whose identity the presented access token names: the DCCF's authenticated subject for the request really ran the service as that NF. (The same claim holds for registration and deregistration.) | `legit_nf` | proved | A02_auth_upd | none |
| A02 | An accepted registration is sent by the NF whose identity the presented access token names: the DCCF's authenticated subject for the request really ran the service as that NF. | `network` | proved | A02_auth | none |
| A02 | The initiator the accepted profile claims (the nwdafId/adrfId in the request body) is an identity that actually ran this service: the DCCF checks that the body identifier is the authenticated consumer. | `legit_nf` | attack (depends_on_ambiguity) | A02_claimed (the control of Z01_upd) | |
| A02 | An accepted update is sent by the NF whose identity the presented access token names: the DCCF's authenticated subject for the request really ran the service as that NF. | `legit_nf` | proved | A02_auth_upd | checked |
| A02 | An accepted registration is sent by the NF whose identity the presented access token names. | `network` | proved | A02_auth | checked |

| target | claim | attacker | answer | query | control |
|---|---|---|---|---|---|
| A03 | The consumer identifier recorded in the accepted profile equals the NF that registered it, so a reader of the profile is assured who registered it. | `legit_nf` | attack (depends_on_ambiguity) | A03_bind (the control of A03_body) | |
| A03 | The collection information stored in the accepted profile is the one the authenticated consumer sent in its request. | `network` | proved | A03_body | checked |
| A08 | The profileId named in an accepted update is the profile of the consumer that sends it: an NF can only update the profiles it registered. | `legit_nf` | attack (holds_in_source) | A08_owner | |
| A08 | An accepted update addresses a profileId that a registration created before it: the DCCF updates an existing profile and never treats an unknown profileId as a profile. | `legit_nf` | bounded (the changing value unrolled to 4 steps) | A08_created_b | none |
| A08 | An accepted update addresses a profileId that a registration created before it. | `network` | bounded (the changing value unrolled to 4 steps) | A08_created_b | checked |
| Z01 | An accepted update is backed by an access token that the NRF issued to the requesting NF after authenticating it; no operation is accepted without such a token. | `legit_nf` | proved | Z01_upd | none |
| Z01 | An accepted registration is backed by an access token that the NRF issued to the requesting NF, and the DCCF rejects a token whose scope and audience are not those of the ndccf-contextmanagement service. | `network` | proved | Z01_reg | none |
| Z01 | An accepted update is backed by an access token that the NRF issued to the requesting NF after authenticating it. | `legit_nf` | proved | Z01_upd | checked |
| Z01 | An accepted registration is backed by a token the NRF issued to that NF, and a token for another service's scope and audience is refused. | `network` | proved | Z01_reg | checked |
| Z03 | A token is bound to the profiles of the consumer it was issued to: it cannot be used to change a profile that carries another consumer's identifier. | `legit_nf` | attack (holds_in_source) | A08_owner | |
| Z03 | No register, update or deregister operation is committed without a token issued by the NRF for the ndccf-contextmanagement service, so a party that has only the network cannot get one committed. | `network` | proved | Z01_upd | checked |
| S05 | A deregistration is accepted by the DCCF, the profile is removed from its store and the consumer receives 204 No Content. | | reachable | run | |
| S05 | A deregistration is accepted by the DCCF and the profile is removed from its store. | `legit_nf` | reachable | run | |
| S05 | A deregistration is accepted only for a profileId the DCCF already holds: the profile removed is the one that was registered. | `network` | attack (model_artifact) | S05_del | |
| A08.7 | The profileId named in an accepted update is the profile of the consumer that registered it, and the recorded nwdafId/adrfId is that consumer's. | `legit_nf` | attack (holds_in_source) | A08_owner | |

| target | claim | attacker | answer | query | control |
|---|---|---|---|---|---|
| A08.7 | An accepted update addresses a profileId that a registration created before it, and its recorded nwdafId/adrfId is the one stored at registration. | `network` | bounded (the changing value unrolled to 4 steps) | A08_created_b | checked |
| Z03.8 | A legitimate NF holding a token for the service may modify or delete only the profiles it registered. | `legit_nf` | attack (holds_in_source) | A08_owner | |
| Z03.8 | A profileId accepted in an update belongs to the consumer that registered it: the DCCF accepts an update only from the NF that created the profile. | `legit_nf` | attack (depends_on_ambiguity) | A08_owner_b (the control of A08_created_b) | |
| Z03.8 | A profile can only be modified or deleted by a party that holds a token for the service; a party on the network alone cannot. | `network` | proved | Z01_upd | checked |
| U01 | The 201 response the consumer receives carries the profileId of the profile created for its own request, and its update and deregistration responses carry the profileId it asked for. | `legit_nf` | proved | A03_body | none |
| U01 | The 201 response the consumer receives carries the profileId of the profile created for its own request, and the lifecycle only completes when the update and deregistration responses carry that same profileId. | `network` | proved | A03_body | checked |
| X01 | The collection information an NF registers reaches only the DCCF it addresses and no other party. | `network` | proved | X01_secret | none |
| X01 | The collection information an NF registers reaches only the DCCF it addresses and no other party, including another legitimate NF. | `legit_nf` | proved | X01_secret | none |

# Appendix D: How the analysis was run

An LLM analyst builds the model of the protocol from the documents and puts each property to ProVerif; every result in section 3 is an answer of the prover.

ProVerif+ Agent session session-20261114T211021, 2026-11-14. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 54, prover 5 s, wall 16 min, cost 0.135 USD.

ProVerif: 89 proofs run, 87 decided, 2 without an answer, 3 s, 31 answers recorded.

## What was analysed

| | |
|---|---|
| **Protocol** | DCCF collection profile lifecycle |
| **Version** | 19.7.0 |
| **Analysed** | Two NFs and one DCCF; aggregate information for the analytics anaSub that does not involve a single UE is selected, and a separate data-collection-profile is registered, updated and deregistered. Actual upstream data collection and transfer between DCCFs are excluded. |
| **Not analysed** | The symbolic model does not prove the computational security or post-quantum security of the algorithms; attack traces hold only under explicit assumptions and are not directly equivalent to specification vulnerabilities. Every property must have a reachability/positive control. Rel-19 is fixed. The external network attacker and the malicious legitimate NF holding its own credentials are distinguished separately; TLS authenticated encryption is modelled per endpoint, and a legitimate NF cannot read another NF's TLS session. OAuth2 subject/audience/scope/expiry and TLS identity binding follow the attached baseline; a service scope does not automatically equal resource ownership. A monotonic table cannot model real deletion, and the finite-state lifecycle must be explicitly bounded. The target is the specified API sub-flow, not full 5G security. |
| **Source text** | 226 numbered passages (31762 bytes) from the sections listed below. A reference such as s0001 in this report is one of these passages |

The sections of the documents the analysis is built from:

| Document | Section | Title |
|---|---|---|
| main | 4.3 | Ndccf_ContextManagement Service |
| main | 4.3.1 | Service Description |
| main | 4.3.1.1 | Overview |
| main | 4.3.1.2 | Service Architecture |
| main | 4.3.1.3 | Network Functions |
| main | 4.3.1.3.1 | Data Collection Coordination Function (DCCF) |
| main | 4.3.1.3.2 | NF Service Consumers |
| main | 4.3.2 | Service Operations |
| main | 4.3.2.1 | Introduction |
| main | 4.3.2.2 | Ndccf_ContextManagement_Register service operation |
| main | 4.3.2.2.1 | General |
| main | 4.3.2.2.2 | Register data collection profile to DCCF |
| main | 4.3.2.3 | Ndccf_ContextManagement_Update service operation |

| Document | Section | Title |
|---|---|---|
| main | 4.3.2.3.1 | General |
| main | 4.3.2.3.2 | Update registered data collection profile |
| main | 4.3.2.4 | Ndccf_ContextManagement_Deregister service operation |
| main | 4.3.2.4.1 | General |
| main | 4.3.2.4.2 | Deregister Data collection profile |
| main | 5.2 | Ndccf_ContextManagement Service API |
| main | 5.2.1 | Introduction |
| main | 5.2.2 | Usage of HTTP |
| main | 5.2.2.1 | General |
| main | 5.2.2.2 | HTTP standard headers |
| main | 5.2.2.2.1 | General |
| main | 5.2.2.2.2 | Content type |
| main | 5.2.2.3 | HTTP custom headers |
| main | 5.2.3 | Resources |
| main | 5.2.3.1 | Overview |
| main | 5.2.3.2 | Resource: DCCF Data Collection Profiles |
| main | 5.2.3.2.1 | Description |
| main | 5.2.3.2.2 | Resource Definition |
| main | 5.2.3.2.3 | Resource Standard Methods |
| main | 5.2.3.2.3.1 | POST |
| main | 5.2.3.2.4 | Resource Custom Operations |
| main | 5.2.3.3 | Resource: Individual DCCF Data Collection Profile |
| main | 5.2.3.3.1 | Description |
| main | 5.2.3.3.2 | Resource Definition |
| main | 5.2.3.3.3 | Resource Standard Methods |
| main | 5.2.3.3.3.1 | PUT |
| main | 5.2.3.3.3.2 | DELETE |
| main | 5.2.3.3.4 | Resource Custom Operations |
| main | 5.2.4 | Custom Operations without associated resources |
| main | 5.2.5 | Notifications |
| main | 5.2.6 | Data Model |
| main | 5.2.6.1 | General |
| main | 5.2.6.2 | Structured data types |
| main | 5.2.6.2.1 | Introduction |
| main | 5.2.6.2.2 | Type: NdccfDataCollectionProfile |
| main | 5.2.6.3 | Simple data types and enumerations |

| Document | Section | Title |
|---|---|---|
| main | 5.2.6.3.1 | Introduction |
| main | 5.2.6.3.2 | Simple data types |
| main | 5.2.6.4 | Data types describing alternative data types or combinations of data types |
| main | 5.2.6.5 | Binary data |
| main | 5.2.7 | Error Handling |
| main | 5.2.7.1 | General |
| main | 5.2.7.2 | Protocol Errors |
| main | 5.2.7.3 | Application Errors |
| main | 5.2.8 | Feature negotiation |
| main | 5.2.9 | Security |

The purpose of the protocol as the source step located it in the documents: two NFs and one DCCF; the selection of aggregated information for analytics anaSub that does not involve a single UE; registration, update and deregistration of an independent data-collection-profile. Excludes actual upstream data collection and transfer across DCCFs.

## The properties against the documents

Before a query was written, the analyst set the drafted list of properties against what the documents themselves say the protocol is for. The name, what a property is about and what the documents say are the analyst's reading; the places named exist in the documents.

| ID | Name | About | What the documents say | Where |
|---|---|---|---|---|
| A02 | Authenticated initiator of each request | The DCCF is assured that an accepted register, update or deregister request was sent by the NF instance (NWDAF or ADRF) that the request claims and that the token names. | The operations are initiated by the NF service consumer (s0026, s0033, s0064, s0079); access may be authorized with OAuth2 client credentials from the NRF (s0223, s0224); NF service request and response shall support mutual authentication between consumer and producer. | ts-33-501:2871, s0026, s0033, s0064, s0079, s0223, s0224 |
| A03 | Registered consumer identifier matches the sender | The consumer identifier stored in the profile (nwdafId, adrfId, nwdafSetId or adrfSetId) together with the collection information is the identity of the NF that registered it, so a reader of the profile is assured who registered it. | The request body shall include one of the identifiers related to the NF service consumer (s0033, s0047 to s0051) and the collection information in anaSub/dataSub; the DCCF creates and stores the profile and returns it (s0052 to s0056). The source does not say that the DCCF compares the body identifier with the authenticated identity. | s0033, s0047, s0052, s0056, s0196 |
| A08 | profileId bound to its creating consumer | The profileId: the DCCF, and the NF that registered, are assured that a profileId addressed by an update or deregistration is the profile that consumer created and still carries the collection information and consumer identifier given at registration. | The DCCF assigns a profileId and returns it in the 201 response and Location header (s0054, s0056); update and deregistration address the profile by {profileId} (s0064, s0079, s0148). The source says nothing about how the profileId is generated or whether it can be guessed. | s0054, s0056, s0064, s0079, s0148 |

| ID | Name | About | What the documents say | Where |
|---|---|---|---|---|
| Z01 | Accepted operations carry a valid service token | The OAuth2 access token: the DCCF accepts a register, update or deregister operation only when the NF presents a token that the NRF issued to that NF for the ndccf-contextmanagement service and for this DCCF. | Access may be authorized with OAuth2 client credentials, NRF as authorization server, and the consumer obtains the token before consuming the service (s0223, s0224); the API defines a single scope ndccf-contextmanagement (s0226); tokens must not be generated or guessed by unauthorized parties. | rfc6749:2870, ts-33-501:2883, s0223, s0224, s0226 |
| Z03 | Token bound to subject, resource and operation | The profileId and the OAuth2 token: a token may only be used for the profiles of the consumer it was issued to and for the operation it authorizes, not for another NF's profile. | The API defines one scope for the whole service with no resource- or operation-level scopes (s0226); RFC 6749 records that the resource server has no method to check that a presented token was issued to the presenting client, and that a malicious client can act for another one when credentials are not kept confidential (rfc6749 10.2, 10.3). The task boundary states that a service scope does not by itself mean ownership of a resource. | rfc6749:2879, rfc6749:2827, s0064, s0079, s0226 |
| S05 | Deregistered profileId no longer valid | The DCCF's profile store: once a deregistration of a profileId is accepted, that profileId no longer identifies a profile and cannot be updated or treated as a valid registration. | Upon an accepted DELETE the DCCF shall remove the corresponding registered profile and answer 204 No Content (s0080, s0081, s0082); the individual profile resource is identified by {profileId} (s0145, s0148). The source does not say whether a profileId may be reused after deletion. | s0079, s0080, s0081, s0082 |
| A08.7 | profileId bound to creator and to nwdafId | The profileId together with the nwdafId/adrfId of the profile: the accepted profileId belongs to the profile this consumer registered, and its recorded consumer identifier is that consumer's. | The DCCF assigns the profileId (s0054) and returns it (s0056); update and deregistration address the profile by {profileId} (s0064, s0079, s0148); the profile body carries nwdafId/adrfId (s0047 to s0051, s0196). The source says nothing about profileId generation. | s0054, s0056, s0064, s0079, s0148, s0196 |
| Z03.8 | Update and deregister authority over a profile | The update and deregister operations on another NF's profile: a legitimate NF holding a token for the service may modify or delete only the profiles it registered. | Update and deregistration are addressed by {profileId} alone (s0064, s0079); the API defines a single service-wide scope with no resource-level scopes (s0226); the task boundary states that a service scope is not resource ownership. | rfc6749:2879, s0064, s0079, s0223, s0226 |
| U01 | Responses match requests and deletion is final | The NF service consumer: the 201/200 response it receives belongs to its own request, and after an accepted deregistration the old profileId is dead. | The DCCF answers 201 with the created profile and a Location header carrying the profile URI (s0056); the PUT answers 200 with the representation or 204 (s0068 to s0070); the DELETE removes the profile and answers 204 (s0080 to s0082). | s0056, s0068, s0069, s0070, s0080, s0081, s0082 |
| X01 | Consumer authenticates the DCCF endpoint | The NF service consumer (NWDAF or ADRF), which holds the token and the profile content, against a party posing as the DCCF. | NF service request and response procedures shall support mutual authentication for direct communication between NF service consumer and NF service producer. | ts-33-501:2871 |

Added from the documents by the analyst: X01.

## The security statements of the documents, paragraph by paragraph

The host listed the 398 paragraphs of the sections in which the documents say what the protocol protects and against what, and the analyst said of each what it made of it: a property, an attacker, or why neither. Taken up: 28. Not taken: 370. Which paragraphs are listed follows from the layout of the standard; where each went is the analyst's judgement.

| Paragraphs | Where | Taken up as | Why not |
|---|---|---|---|
| P001 to P002 | rfc6749:2778-2786 | No goal stated | It says that OAuth security considerations depend on the client profile and points at the OAuth threat model; nothing about this API's properties. |
| P003 to P006 | rfc6749:2790-2823 | Z01, `legit_nf` | |
| P007 to P010 | rfc6749:2827-2850 | Z01, Z03, `legit_nf` | |
| P011 to P015 | rfc6749:2859-2881 | Z01, Z03, `network`, `legit_nf` | |
| P016 to P021 | rfc6749:2885-2917 | No goal stated | Refresh tokens do not exist in the client-credentials grant this API uses (s0223). |
| P022 to P025 | rfc6749:2921-2943 | No goal stated | The authorization-code grant and its redirection endpoint are not used: the consumer uses the client-credentials grant (s0223). |
| P026 to P029 | rfc6749:2947-2980 | No goal stated | No redirection URI is registered or used in the client-credentials grant. |
| P030 to P033 | rfc6749:2984-3003 | No goal stated | The resource-owner password grant is not used; there is no resource owner in this API. |
| P034 to P035 | rfc6749:3013-3019 | Z01, `network` | |
| P036 | rfc6749:3023-3028 | X01, Z01, `network` | |
| P037 to P039 | rfc6749:3032-3041 | The prover cannot be asked | The probability of guessing a generated token (2^-128) is a property of token generation and entropy, which a symbolic prover cannot be asked. |
| P040 to P042 | rfc6749:3045-3066 | No goal stated | Phishing of end users; this API has no end user. |
| P043 to P048 | rfc6749:3070-3110 | No goal stated | CSRF protection concerns the authorization endpoint and the user-agent; the client-credentials grant has neither. |
| P049 to P050 | rfc6749:3119-3138 | No goal stated | Clickjacking concerns an authorization page shown to an end user; there is none in this API. |
| P051 to P052 | rfc6749:3142-3150 | No goal stated | The values named (state, redirect_uri) do not exist in the client-credentials grant; body validation of this API is covered by A03 and A08. |
| P053 to P054 | rfc6749:3154-3167 | No goal stated | Open redirectors concern redirection endpoints, which this grant does not use. |
| P055 to P062 | rfc6749:3170-3210 | No goal stated | The implicit flow, its token substitution and audience restriction are not used: the consumer obtains its token over the back channel by client credentials (s0224). |
| P063 | ts-33-501:2491-2491 | Outside the scope | Bidding down of the UE's security features, not the SBI sub-procedure modelled here. |
| P064 to P072 | ts-33-501:2495-2511 | Outside the scope | Requirements on the authentication of the UE's subscription and of the serving network; the task excludes per-UE procedures. |
| P073 to P074 | ts-33-501:2515-2517 | Outside the scope | Requirements on 5GC and NG-RAN keys for AS and NAS protection. |
| P075 to P078 | ts-33-501:2523-2529 | Outside the scope | UE ciphering and integrity with the ng-eNB over Uu. |
| P079 to P089 | ts-33-501:2533-2553 | Outside the scope | User data and signalling data confidentiality over the radio and N3 interfaces. |

| Paragraphs | Where | Taken up as | Why not |
|---|---|---|---|
| P090 to P103 | ts-33-501:2557-2583 | Outside the scope | User data and signalling data integrity over the radio and N3 interfaces. |
| P104 to P110 | ts-33-501:2587-2599 | Outside the scope | Secure storage and processing of subscriber credentials in the UE, USIM and network functions, not of the DCCF's profiles. |
| P111 to P125 | ts-33-501:2603-2631 | Outside the scope | Subscriber privacy; the analysed sub-procedure selects aggregated analytics that do not concern a single UE. |
| P126 to P213 | ts-33-501:2637-2853 | Outside the scope | UE, NG-RAN, gNB, AUSF, UDM and UDR requirements for the access network and subscriber procedures; the task covers only the DCCF profile lifecycle over the SBI. |
| P214 | ts-33-501:2859-2859 | Z01, X01, `network` | |
| P215 | ts-33-501:2865-2865 | No goal stated | Replay protection for service discovery and registration is a property of the SBI transport that the endpoint-channel model carries, not a message-level property of the profile lifecycle; the lifecycle's own replay question is asked by S05 and U01. |
| P216 | ts-33-501:2867-2867 | Z01 | |
| P217 | ts-33-501:2869-2869 | Outside the scope | Topology hiding between administrative domains belongs to SEPP and roaming, which the task leaves out. |
| P218 | ts-33-501:2871-2871 | A02, X01, `network`, `legit_nf` | |
| P219 | ts-33-501:2873-2873 | A02, `legit_nf` | |
| P220 | ts-33-501:2875-2875 | A03, Z01, A08 | |
| P221 to P227 | ts-33-501:2879-2891 | Z01, A02, `network`, `legit_nf` | |
| P228 to P234 | ts-33-501:2895-2907 | Outside the scope | Requirements on the NEF and the AF interface; the analysed sub-procedure has no external exposure. |
| P235 to P241 | ts-33-501:2911-2923 | Outside the scope | Requirements on the SCP for indirect communication, which the task leaves out. |
| P242 to P312 | ts-33-501:2929-3075 | Outside the scope | End-to-end interconnection security with SEPP, roaming intermediaries, protection of attributes and IPUPS: roaming and inter-PLMN communication are outside the modelled sub-procedure. |
| P313 to P314 | ts-33-501:3079-3081 | Outside the scope | IPUPS requirements for interconnect, outside the modelled sub-procedure. |
| P315 to P318 | ts-33-501:3085-3091 | Outside the scope | The 3gpp-Sbi-Originating-Network-Id header and PLMN ID handling concern inter-PLMN signalling. |
| P319 to P337 | ts-33-501:3097-3141 | Outside the scope | Requirements on signalling monitoring, security visibility and configurability of the operator's network, not on this API's messages. |
| P338 to P369 | ts-33-501:3149-3215 | Outside the scope | Ciphering and integrity algorithm identifiers and algorithm selection for the access network; the strength of a primitive is outside a symbolic model. |
| P370 to P380 | ts-33-501:3219-6548 | Outside the scope | Requirements on 5G-RG, NSSAAF and N3, which are not part of the DCCF profile lifecycle. |
| P381 to P398 | ts-33-501:9656-11901 | Outside the scope | Annexes on EAP-TLS, IAB, MBSF/MBSTF and NEF requirements, outside the modelled sub-procedure. |

## The parties and the protected values, one by one

The host listed the 13 parties and protected values that the drafted properties name, and the analyst said of each which attacker controls the party or holds the value, or why no such attacker was set. With an attacker: 2. Without: 11. The list follows from the properties; what was made of each entry is the analyst's judgement.

| ID | Party or value | Kind | Attacker | Why none |
|---|---|---|---|---|
| S01 | DCCF | party | Everything lost by definition | The DCCF is the party that performs every check and keeps the store. A controlled DCCF makes all nine targets void by definition, and the documents ask none of them under a dishonest DCCF. |
| S02 | NWDAF consumer | party | `legit_nf` | |
| S03 | ADRF consumer | party | `legit_nf` | |
| S04 | NRF authorization server | party | Everything lost by definition | With the authorization server controlled, tokens can be minted for any subject, so every token-based property (A02, Z01, Z03) is void by definition. The profile-store properties (A08, S05) do not depend on the NRF and are asked for the network and legitimate-NF attackers instead. |
| S05 | profile ownership identity | value | Nothing gained | The profile's ownership identity is what the DCCF records from the request body. A party reading it gains nothing beyond sending a request, which is the legitimate-NF attacker already. |
| S06 | collection information | value | Nothing gained | The collection information is carried to the DCCF over the NF's own endpoint channel. A party that reads it holds the NF's credentials, which is the legitimate-NF attacker. |
| S07 | consumer identifier | value | Nothing gained | The consumer identifiers nwdafId/adrfId are public identities of the NF instances. |
| S08 | profileId | value | Outside the scope | Whether a profileId can be guessed depends on how the DCCF generates it, which the source does not specify. The analysis covers both readings by asking A08 on the model with a fresh identifier and on the unrolled model, and guessing itself is an entropy question. |
| S09 | OAuth2 access token | value | Nothing gained | A token is only usable from the NF's own authenticated endpoint channel, so a party holding a token but no NF credentials gains nothing. A party holding both is the legitimate-NF attacker. |
| S10 | profile resource | value | Nothing gained | The profile resource lives at the DCCF and is reachable only through the checks modelled. The way to reach it is to be a legitimate NF, which is the attacker already defined. |
| S11 | profile contents | value | Nothing gained | Profile contents are the collection information a consumer chooses and sends itself. A party holding them holds what the registering NF sent. |
| S12 | nwdafId/adrfId | value | Nothing gained | nwdafId/adrfId are public NF instance and set identifiers used for discovery. |
| S13 | response | value | Nothing gained | A response carries the profile representation and the profileId back to the NF that made the request. A party that reads it is on that NF's endpoint channel. |

## The properties under each attacker

The attackers are the analyst's, set before a query was written; each answer was recorded under one of them. A cell is the prover's answer for the property under that attacker; "No answer" is an attacker set for the property under which the prover left a claim undecided or nothing was recorded; an empty cell is an attacker the analyst did not set for the property.

| Attacker | Has | Basis | Where |
|---|---|---|---|
| `network` | the network only: it reads, drops, changes and sends messages, and holds no secret of a party | | |
| `legit_nf` | The client credentials of a genuine NF service consumer instance (an ADRF or NWDAF in the same network) and, with them, a valid access token for the ndccf-contextmanagement scope issued by the NRF; it speaks HTTP/2 over its own authenticated TLS connection to the DCCF, chooses the consumer identifier and collection information it puts into a request body, and names any profileId it can construct or guess. | The task boundary separates the external network attacker from a malicious legitimate NF that holds its own credentials; consumers of this service are NWDAF and ADRF (s0010, s0011), the API has one service-wide scope (s0226). | rfc6749:2827, s0010, s0011, s0223, s0226 |

| ID | `network` | `legit_nf` |
|---|---|---|
| A02 | Holds | Fails |
| A03 | Holds | Fails |
| A08 | Holds | Fails |
| Z01 | Holds | Holds |
| Z03 | Holds | Fails |
| S05 | No answer | No answer |
| A08.7 | Holds | Fails |
| Z03.8 | Holds | Fails |
| U01 | Holds | Holds |
| X01 | Holds | Holds |

## Source and targets

Source: 226 units, 31762 bytes, frozen in source-delivery-01.json. Targets: 7 approved (A02, A03, A08, Z01, Z03, S05, U01); the questions asked: 3. The list of targets was approved as drafted, beforehand, with --approve-targets.

Decisions the host took by its documented default (task `decisions: auto`), for you to check:

- approve_package: approved source package protocol-ddaaa5507937a0ef8f8ff266ee1e

This step as it reports itself: 2 requests (2 failed), 0.000 USD, 68074 tokens in and 7953 out, 222 s.

## Notes on the prover and the model

- A08: answered in ProVerif, and the analyst names changing_state for this target
- Z03: answered in ProVerif, and the analyst names changing_state for this target
- S05: answered in ProVerif, and the analyst names changing_state for this target
- A08.7: answered in ProVerif, and the analyst names changing_state for this target
- Z03.8: answered in ProVerif, and the analyst names changing_state for this target
- U01: answered in ProVerif, and the analyst names changing_state for this target

A ProVerif model that simplifies algebra (xor, dh_algebra, other_algebra) gives the attacker less than the real operation: a proof there is a proof against a weaker attacker, an attack there is an attack. A ProVerif model that leaves out a check on a value that changes (changing_state) lets parties accept more: a proof there holds, an attack there may be nothing but the missing check. Tamarin+ Agent models these as they are.

A bound in steps is an answer on bounded.pv: the analyst's second ProVerif model, in which the party that keeps a value that changes is unrolled to that many steps with its check in place. That model does less than the protocol does, so the property holds for those steps and no further, and an attack found there is an attack. The number of steps is the analyst's statement.

## State review by the analyst

This part is the analyst's own reading of the source, not an answer of the prover. "yes" under shown by ProVerif means that ProVerif found a trace with these steps in the unrolled model, so the steps are possible; that they break a target is said only by an attack in the table of answers, and what else they cost is the analyst's reading. "nothing found" says which patterns were looked at and proves nothing.

| value | kept by | changes when | checked when | refs |
|---|---|---|---|---|
| profileId and the stored data collection profile (consumer identifier, collection information) | DCCF | a registration creates it, an update replaces its contents, a deregistration removes it | an update or a deregistration names it: the DCCF must find the stored profile for that profileId | s0052, s0053, s0054, s0055, s0064, s0065, s0066, s0067, s0079, s0080, s0081, s0082, s0209, s0214 |
| OAuth2 access token for the ndccf-contextmanagement scope | NF service consumer (NWDAF, ADRF) | the NF obtains a new token from the NRF before consuming the service | on every register, update and deregister request the DCCF validates the token's signature, scope and audience | s0223, s0224, s0226 |

| pattern | result | target | shown by ProVerif | steps |
|---|---|---|---|---|
| replay | candidate | | no: not tried with ProVerif | The NF service consumer sends POST /data-collection-profiles with a valid token (s0033). / The DCCF creates a new profile, assigns a profileId and stores it (s0052 to s0055). / The same POST is sent again with the same token; the DCCF has no rule to recognise the repetition and creates a second profile with a second profileId (s0052 to s0055). / Two profiles now exist for one registration intent; neither the source nor the model rules this out. |
| repeat_one_time_action | candidate | | no: not tried with ProVerif | A consumer repeats its registration request because the 201 response was lost. / The DCCF creates a further profile for the same collection information (s0053 to s0055). / Registration is therefore not a one-time action per consumer and collection information. |
| update_before_check | candidate | | no: not tried with ProVerif | A consumer sends PUT /data-collection-profiles/{profileId} naming a profileId that does not exist (e.g. one that was deregistered, or one the consumer guessed). / The DCCF is told to update the profile of the corresponding profileId and to store the profile (s0065 to s0067); the source names no existence check and no 404 response for this case. / The DCCF either refuses with an error (s0071) or stores a profile under that profileId; the source does not say which. |
| race_between_sessions | candidate | Z03.8 | yes: ProVerif falsifies `A08_owner_b` on bounded.pv | The NWDAF registers a profile and receives its profileId (s0054, s0056). / The ADRF, holding its own valid token for the same service scope, sends PUT on that profileId (s0064). / The DCCF checks the token but not who registered the profile and replaces the profile contents, so the NWDAF's registration now says what the ADRF sent (s0065 to s0067). / Nothing in the source orders the two writers or binds the profileId to its registrant. |

| pattern | result | target | shown by ProVerif | steps |
|---|---|---|---|---|
| distinguishable_failure | candidate | | no: not tried with ProVerif | A consumer sends PUT or DELETE for a profileId it does not own. / The DCCF answers with an HTTP error; the source leaves the choice of error to clause 5.2.7 (s0071, s0083) and lists application errors separately (s0214 to s0217). / If the error for 'no such profileId' differs from the error for 'not your profile', a consumer learns from the difference whether a profileId exists, i.e. how other NFs' profiles are numbered. |
| no_rollback_on_failure | nothing found | | | The source says that on an error the DCCF sends an error response (s0071, s0083) and says nothing about a partially applied update, so no value is left changed; the unrolled model applies an update only after the token checks. |
| desynchronisation | nothing found | | | The consumer holds no copy of the profile beyond the response it received; after an accepted deregistration both sides consider the profileId gone, and the source gives no recovery step that would let them diverge. |
| wrap_or_window | nothing found | | | The profile lifecycle keeps no counter or acceptance window; the profileId is assigned, not incremented by the consumer, and the source sets no window. |

What the analyst named for targets:

| step | targets | features | reason |
|---|---|---|---|
| 36 | A08, A08.7, S05, U01, Z03, Z03.8 | changing_state, many_sessions | The DCCF's stored profile table and the profileId it assigns change across the lifecycle: registration creates the profile, update replaces its contents, deregistration removes it, and the profileId is a value the DCCF hands out and that other parties may try to name. A ProVerif table only grows, so these properties are asked on model.pv for any number of sessions and on bounded.pv, where the store is threaded through an unrolled chain of four steps with real removal. Answers that rest on the profileId being predictable come from the unrolled model, where the DCCF hands the profileId out in sequence. |

## The analyst's closing note

Model and answers are recorded. The plain run exists in model.pv (register, update, deregister over TLS-modelled endpoint channels with an NRF-issued token), and every target has an answer under both attackers (network alone, and a legitimate NF with its own credentials).

What holds: every committed register, update or deregister is backed by an NRF-issued token for the ndccf-contextmanagement scope and this DCCF (Z01, proved, with control), the authenticated subject really ran the service (A02, proved), the stored collection information is what the authentic consumer sent (A03, proved), a profile is only updated when the DCCF holds its profileId (A08, bounded on 4 unrolled steps, with control), and the collection information reaches only the DCCF (X01, proved).

What fails, and is the finding: the DCCF never compares the consumer identifier in the request body with the subject of the token, and never checks who registered a profileId. A second legitimate NF therefore registers a profile that says nwdafId = NWDAF (attack on A03, A02, A08.7), and — with the profileId known or predictable, which the source leaves entirely open — updates or deregisters another NF's profile (attacks on A08, Z03, Z03.8, and a confirmed race in the state review). Novelty is unclear: the documents state one service-wide scope and no ownership rule, but never state that this breaks the binding.

What blocks the rest: ProVerif's monotone profile table cannot model removal and produces falsifications for the correspondence over the table, and it cannot reconstruct a trace for the later steps of the unrolled chain, so S05's post-deletion claim and U01's state half are recorded as reachable/model-artifact and stay open. Tamarin+ Agent, which models the stored profile and its removal as they are, would settle them. Recommendations (compare the body identifier with the token subject; bind the profile to its registrant and reject a PUT/DELETE from another subject with 403) were not machine-checked, because the fix is a missing check rather than a change of messages.