RCS Extensibility Terminal API Security

Version 1.0

15 October 2014

This is a Non-binding Permanent Reference Document of the GSMA

Table of Contents

1 Introduction

1.1 Overview

This note outlines a proposal to support extensible RCS (Rich Communication Services), looking at security issues for the API and stack, and how those extensions are managed.

NOTE 1: The approach set out in this document is in response to a request for a "federated" security approach without a central approver of apps. A completely federated approach that meets all business requirements has not been achievable but a balance between MNO autonomy and interoperability with a minimum of central control is proposed.

NOTE 2: The proposal in this document seeks to provide adequate identification and security mechanisms and processes to facilitate MNO-developed applications using Terminal APIs and to have a smooth process of expansion access to Terminal APIs to third party ISVs (Independent Software Vendor) in the future without having to re-engineer the security mechanisms. Expansion to third party ISVs will entail a significant amount of process development and software tool provision on each developer's part, which is out of scope of this document.

NOTE 3: This document relates to the RCS 5.2 Specification and not any earlier versions of the RCS specification.

1.2 Document Structure

Section 2 contains an introduction to the objectives and scope of this document and the use cases supported by this approach.

Section 3 describes the approach to terminal API access control. This identifies the methods by which applications gain access to sensitive APIs.

Section 4 sets out the different situations in which RCS extensions might be created. Three principal use cases are discussed and their high-level requirements are stated.

Section 5 outlines the process for managing MNO-developed extensions.

Section 6 outlines the process for managing third-party ISV-developed extensions and for controlling their corresponding tags.

Section 7 identifies the concrete steps needed to implement the proposal, addressing additions to RCS specifications and terminal and infrastructure implementations.

1.3 Definitions

Term	Description
App	That means mobile application in this document
RCS extensions	Applications adding functionality to native devices utilising RCS APIs

1.4 Abbreviations

Term	Description
CA	Certificate Authority
CRL	Certificate Revocation List
DO	Developer Operator
IARI	IMS Application Reference Identifier (IARI)
ICSI	IMS Communication Service Identifier
ISV	Independent Software Vendor
MNO	Mobile network operator
MSRP	Message Session Relay Protoco
OCSP	Online Certificate Status Protocol
OEM	Original equipment manufacturer
RCS	Rich Communications services
SAN	Subject Alternative Name
SIP	Session Initiation Protocol
SO	Serving Operator
UI	User interface
UX	User experience

1.5 References

Ref	Doc Number	Title
[1]	[RCS5.2]	GSMA PRD RCC.07 - RCS 5.2 - Advanced Communications: Services and Client Specification, Version 5.0, 07 May 2014 http://www.gsma.com/rcs/
[2]	RFC 2119	“Key words for use in RFCs to Indicate Requirement Levels”, S. Bradner, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

1.6 Conventions

The key words “must”, “must not”, “required”, “shall”, “shall not”, “should”, “should not”, “recommended”, “may”, and “optional” in this document are to be interpreted as described in RFC2119 [2].

2 RCS Extensibility and Terminal API Security

2.1 Context

As part of the Network 2020 Programme, the GSMA is supporting member operators in launching commercial deployments of RCS, enabling applications and services running on devices with native RCS stacks to be delivered over the IP Multimedia Subsystem (IMS) infrastructure. The RCS 5.2 specification defines a range of services including enriched call, messaging, multimedia and other services, which are natively available on RCS-enables handsets.

However, operators require additional services to be created and deployed in an agile fashion without having to be synchronised with the cycle of new specifications and new devices. For RCS to compete as a platform, in the face of the multitude of new applications and services being delivered independently over IP, it is important to be able to create new services outside of the scope of pan-network standards. Such services might be experimental (targeting eventual standardisation), or MNO-specific (associated with an MNO's own service offering), or might be entirely private to a single application.

This approach is enabled by exposing Terminal RCS Services APIs on the handset and deploying applications, initially created by Terminal Device OEMs and MNOs and eventually sourced from third party ISVs.

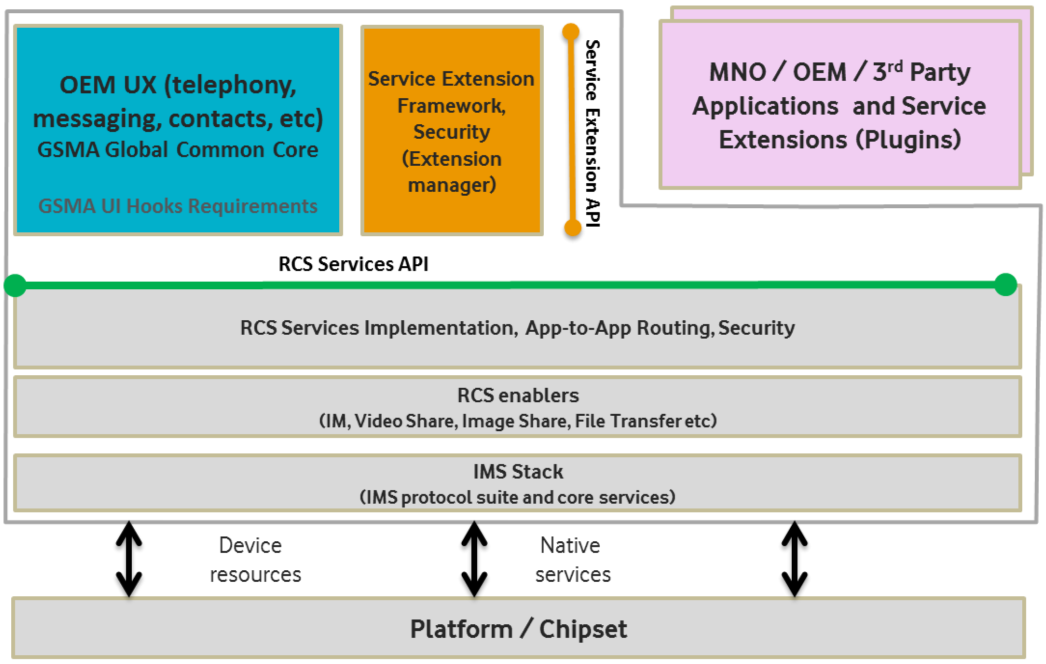

OEM UX (telephony, messaging, contacts, etc)
GSMA Global Common Core
GSMA UI Hooks Requirements
Service Extension Framework, Security (Extension manager)
Service Extension API
MNO / OEM / 3rd Party Applications and Service Extensions (Plugins)
RCS Services API
RCS Services Implementation, App-to-App Routing, Security
RCS enablers
(IM, Video Share, Image Share, File Transfer etc)
IMS Stack
(IMS protocol suite and core services)
Device resources
Native services
Platform / Chipset

Figure 1: General Architecture Overview

The underlying RCS protocols (chat, file transfer, etc.) are extensible. Each existing application or service is assigned a unique name - formally an IMS Application Reference Identifier (IARI) – but commonly referred to simply as a tag. New services can also be defined and, once they are assigned a tag, their traffic can also be carried over the network. The tag, like an application-specific "label" on all of an app's network traffic, allows that traffic to be handled appropriately in the network and also routed correctly to the application on each device. Tags are also the basis on which one device can discover which applications or services are available on its known peers. This extensibility means that later revisions of the RCS standards can incorporate new services as they are developed.

The current generation of social messaging apps, for example, shows the potential of this approach; instead of seeking a single service proposition that meets all of the needs of the market, there can be multiple different services existing to satisfy the niche needs and interests of different groups. RCS should be able to act as the enabling infrastructure for such services instead of being a constraint by prescribing a single, frozen, feature set.

The most significant challenges in enabling this extensibility in practice are;

- management of applications and their tags for discoverability and traffic routing
- ensuring globally unique tags are reliably associated with the providers of associated applications
- enablement of a future developer ecosystem
- meeting security considerations in a non-intrusive way.

These issues and proposed solutions are the subject of this document.

2.2 Contents

Section 2 contains context, use cases and details of what is and is not enabled by the proposed approach to API.

Section 3 describes the approach to identification of applications.

Section 4 sets out the different situations in which extensions might be created. Three principal use cases are discussed and their high-level requirements are stated.

Section 5 outlines the process for managing MNO-originated extensions.

Section 6 outlines the process for managing third-party extensions and for controlling their corresponding tags.

Section 7 identifies the concrete steps needed to implement the proposal, addressing additions to RCS specifications and terminal and infrastructure implementations.

2.3 Scope

2.3.1 What is enabled by the proposed approach

API access:

- Access by system (Terminal Device OEM) apps to all RCS terminal APIs;
- Access by any app to “Open” APIs that are considered to be non-sensitive, or of low sensitivity such that the user can authorise access. (see 4.3.1)
- “Controlled openness”: access by applications to “Controlled” APIs that are considered to be more sensitive, subject to independent and ongoing application authorisation. (see 4.3.2)

Core service APIs:

- Enablement of apps using the Core Services APIs and which interoperate with the native UI for core services on other RCS-enabled devices.
- Control by the service provider (MNO or MVNO) over which applications can use controlled APIs including core service APIs on the devices subscribed to their network.
- Revocation by the service provider (MNO) of access by specific applications to sensitive core services APIs.

RCS extensions:

- Enablement of MNO-specific applications which are assigned custom feature tags either by the MNO or by a developer empowered to do so by a MNO, and which only operate on devices carrying a certificate from that MNO.
- Enablement of third-party applications for which unique custom feature tags are created by the developer.
- Identification of applications on a device implementing both MNO-specific and third-party RCS extensions during Capability Discovery;
- Identification of the traffic pertaining to a specific RCS extension, for app-to-app traffic routing, for both MNO-specific and third-party extensions.
- Prevention of apps using APIs without getting prior approval from the MNO.
- Security mechanism to prevent developers spoofing or forging an MNO-specific feature tag
- Prevention of apps from using the unique custom feature tag belonging to any other third party without getting prior approval from the creator of that tag.
- Revocation by an individual service provider (MNO) of access by specific applications to sensitive RCS extension APIs for devices subscribed to that provider's network.
- Global revocation of access by specific applications to sensitive RCS extension APIs across all networks.

NOTE: The assumption is this process can in future scale to support full 3rd party access to T-APIs. The workflow and processes supporting that are out of scope.

2.3.2 What is not enabled by this approach

- Preventing an app on a jail-broken or hacked device from using an MNO-specific custom tag or another third-party app's unique custom tag.
- Revocation of apps in the sense of removing them from end users devices is not supported.
- Differentiated access to APIs; an application, if authorised to use controlled core service APIs, can use all core service APIs
- Ability to retrospectively add a new core service to the stack. (NOTE: A proposed approach may be submitted as a CR for a possible maintenance release of RCS 5.2)

2.4 Supported Use Cases

Case #	Title	Description
1	Service Integration by Device OEM at manufacture	The OEM will integrate the RCS stack and conforming API implementation, as well as the app accessing the API. The natively integrated apps – whether for core services or for any MNO-specific or other RCS extensions – are required to be able to access the terminal API.
2	Service Addition by end user	Users can install an app that offers an RCS-based communication service, using on-device RCS stack, coexisting with existing apps Apps can send messages to the same app on other devices MNOs can choose to allow any ISV to create such an app OR

Case #	Title	Description
		only allow ISVs approved by them OR not allow any apps MNOs can block traffic on their own network from an app without affecting core services from native devices or other apps
3	User Interface replacement by end user	Users can install an app that offers an RCS-based communication service, using on-device RCS stacks, coexisting with existing apps Apps can send messages to the native app for that traffic type on other devices MNOs can block traffic on their own network from such an app without affecting core services from native devices or other apps
4	Service Extension	Entry points to installed RCS apps can appear to the end user directly in the screens in which they would naturally need them e.g. dialler, message composer, address-book, contact card (“headless apps”)
5	Service discoverability [App discoverability]	Installed apps are discoverable to the end users contacts using the RCS capability discoverability method (such discoverability can be filtered by a MNO)
6	App to app traffic routing	A message sent from an RCS-enabled app or service extension can be received as in the RCS inbox OR directly in the B-party’s app Every app is identified by own globally unique IARI Traffic can be blocked on network by IARI Apps can be blocked from accessing TAPI on device by IARI ISVs can add discoverability and in-app communications to their own non-communications apps for viral distribution, recommendation-based distribution, social gaming, user-to-developer comms.
7	API access control & revocation	MNOs can choose (at registration) whether only apps with a tag from a given tag range AND/OR Apps with tag from MNO’s own tag range AND/OR Apps from a tag range agreed between specific Operators AND/OR Apps from any ISV may access the APIs. MNOs can verify at the network whether a specific IARI is associated with an approved app (e.g. one that has signed its terms and conditions) before supporting traffic from the app MNOs can revoke access to the APIs for a specific application for all devices

Case #	Title	Description
		MNOs can block traffic by IARI (at IMS/application gateway)
8	Download communications app, RCS as a bearer	Third parties can develop their communications apps to run on RCS rather than OTT IP communications , subject to those apps being given permission to use necessary APIs
9	Adherence to Terms and conditions	The MNO can request the 3rd party developer to sign up to their terms and conditions as a pre-requirement before being granted access to APIs (procedural; means app will be blocked if not in adherence) The MNO can verify whether a specific IARI is associated with an approved app (e.g. one that has signed its terms and conditions) before supporting traffic from the app
10	Controlled discoverability	The end user can see from within an app and the contact card which of their contacts have the same app and is prompted at app install to select whether the app should be discoverable or not. The end user can opt to have an RCS app not be discoverable. The end user cannot see all the apps on the B parties device, just the ones they have in common The end user sees apps which may only be installed on the B party's secondary devices The end user apps are no longer discoverable once deleted
11	Usage tracking	The MNO shall be able to discover how many instances of each app are active on their network.
12	Traffic measurement	The MNO shall be able to deploy a network element to measure the IARI-specific traffic.

Table 1: Supported Use Cases

3 Application identification

3.1 Introduction

This section summarises the mechanisms for application identification, which underpin the other mechanisms described in the later sections.

IARIs are used for four specific purposes. The first is to make apps discoverable. The IARIs of applications on a device are returned as part of Capability discovery, allowing peer apps on the device triggering the capability discovery to know that the contact queried is not only RCS capable but has the same app on their device.

The second purpose is to identify traffic from a specific application to the RCS stack and to the network. The application identifier is in the Contact header of all network traffic generated by the application; this then allows usage (number of applications on the network, amount of traffic they generate) to be monitored. A third-party chat client, for example, can generate chat traffic, communicating with other chat clients, but its traffic is separately identifiable in the network and stack.

The third use of application identifiers is to enable peer apps to create private “virtual networks” in which RCS services are used privately between two identical applications or peers, on different devices. Those peers, once authorised, can perform private application-to-application communication without interference from other apps. This can support in-game communications, the transfer of application traffic using RCS as a bearer, and/or communication between the app or game developers and users for maintenance or customer care.

Such private app-app traffic can, in principle, use any protocol supported by the RCS specification and by the device stack; this might be a datagram-like service based on SIP MESSAGE or a connection-oriented service based on MSRP. It is even possible that an extension might reuse the functionality of an existing core service (such as group chat) but, using a custom identifier, operate distinctly and privately between peer extension instances.

NOTE: Some apps are designed to replace the native messenger UI or may be designated the default messaging app on the device by the end user. These apps also have an IARI but their traffic is routed to the default UI for that communication type on the B-party device, which may by the native UI the same app or another app. A description of how this is supported is set out in section 3.5.

The final purpose of application identifiers is to allow, when circumstances demand, the traffic from that application to be blocked in the network. The IARI range(s) approved by an operator are also included in a certificate passed to the device on provisioning (and possibly updated via Device Management) which indicates to the stack on the device which IARIs may access the controlled RCS APIs on the stack. This prevents an unapproved or unknown or blocked app from even accessing the APIs or generating RCS traffic.

The processes for assigning and managing these identifiers (IARIs) are described below.

3.2 IARI structure

An IARI or tag for RCS has the general form:

```
urn:urn-7:3gpp-application.ims.iari.xxx
```

where xxx can be the name of an individual service, or can be a structured range of names.

A registry of standardised services and their registered IARIs1 is maintained by 3GPP for the core services. Examples of standard services are:

- the GSMA Image Share service:

  ```
  urn:urn-7:3gpp-application.ims.iari.gsma-is
  ```

- the RCS IM Chat service:

[1] http://www.3gpp.org/specifications-groups/34-uniform-resource-name-urn-list

```
urn:urn-7:3gpp-application.ims.iari.rcse.im
```

Within this structure, there are ranges of tags whose management may be delegated to other organisations. Each participating MNO has a range of tags that they can manage themselves, which is based on their Mobile Country Code (MCC) and Mobile Network Code (MNC); for example any tag of the form:

```
urn:urn-7:3gpp-application.ims.iari.rcs.mnc<mnc>.mcc<mcc>.<service>
```

is managed by the MNO owning the given MCC/MNC.

As an example, a photo-sharing app deployed by Orange on its network might have the tag:

```
urn:urn-7:3gpp-application.ims.iari.rcs.mnc33.mcc234.photo-share
```

3.3 Second-party applications

This case covers applications that are created by an MNO, or a party acting on the MNO's behalf, and the application is assigned a tag within the MNO's delegated range of names. The MNO may manage the tag names within its delegated range any way it chooses; it also exclusively decides which apps may use those tag names. Once a given application is authorised to use a specific tag, that authorisation is captured as a digital signature that is packaged as part of the application. Verification of that signature, confirming to the stack that the application is authorised to use that tag, is performed using a root certificate known to be associated with that range based on the stack's configuration data. The process is summarised in the figure below.

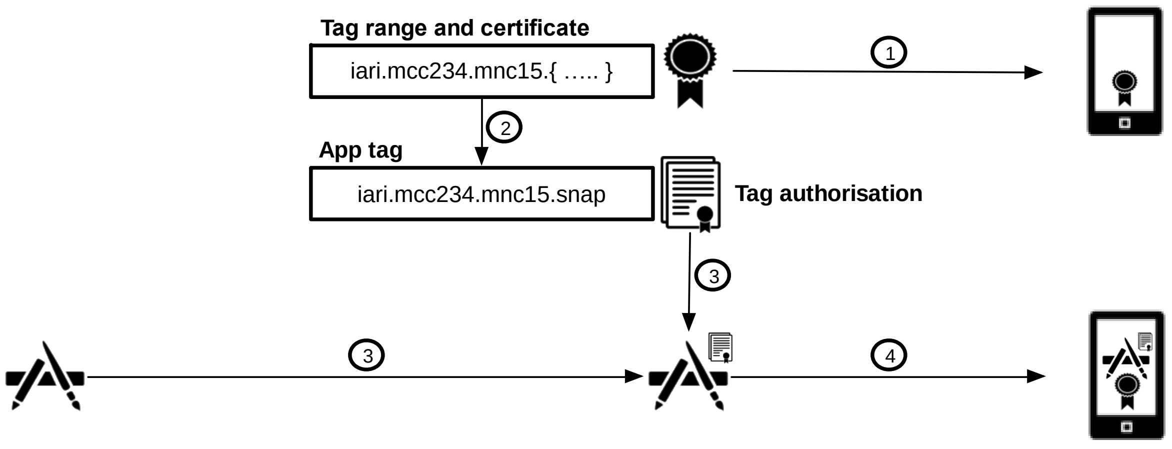


Figure 2: Second-party applications

1. Tag range owner releases the tag range certificate, which is configured with the RCS stack on each device that supports that range.
2. The tag range owner creates a specific tag for an app, and creates a signed Tag Authorisation associating that tag with that app.
3. The IARI Authorisation is packaged as part of the app by the application developer or publisher.
4. Upon installation, the RCS stack verifies that the app is authorised to use that tag.

More generally, any owner of a “Tag Range” can follow the same procedure; for example there may be a role for a new delegated space of identifiers for pre-standard services that is MNO-neutral. These would nonetheless be treated, from a security point of view, as second-party extensions; they provide something that is part of the offering of the service provider, and arbitrary third-party apps are not permitted to use their identifier.

The validated tag for an application is known by the stack and is used as the IARI in the headers of SIP traffic generated by the application. This can be used for application discovery, for the establishment of private app-to-app virtual communications, and also to mediate access to the service, or to specific sensitive APIs, depending on the access control policy of the service provider.

3.4 Third-party applications

This case covers applications developed independently by a third party. Third-party applications are supportable in principle without the need for any central issuer of identities (for developers or their apps) and without any prerequisite for independent application evaluation or scrutiny. Third-party developers may then freely create and distribute apps that use RCS with this identity to communicate, and be discoverable peer-to-peer, without any independent assurance of their identity.

Third-party application use tags in a specific range, having the format:

```
urn:urn-7:3gpp-application.ims.iari.rcs.ext.ss<app-specific string>
```

The application-specific string is independently generated and unique. As with the second-party case, a Tag Authorisation is generated that binds a specific app to that tag; this is performed using a cryptographic mechanism that assures that only the creator of the unique tag string (the “tag owner”) could have authorised the app.

The process is illustrated in the following figure.

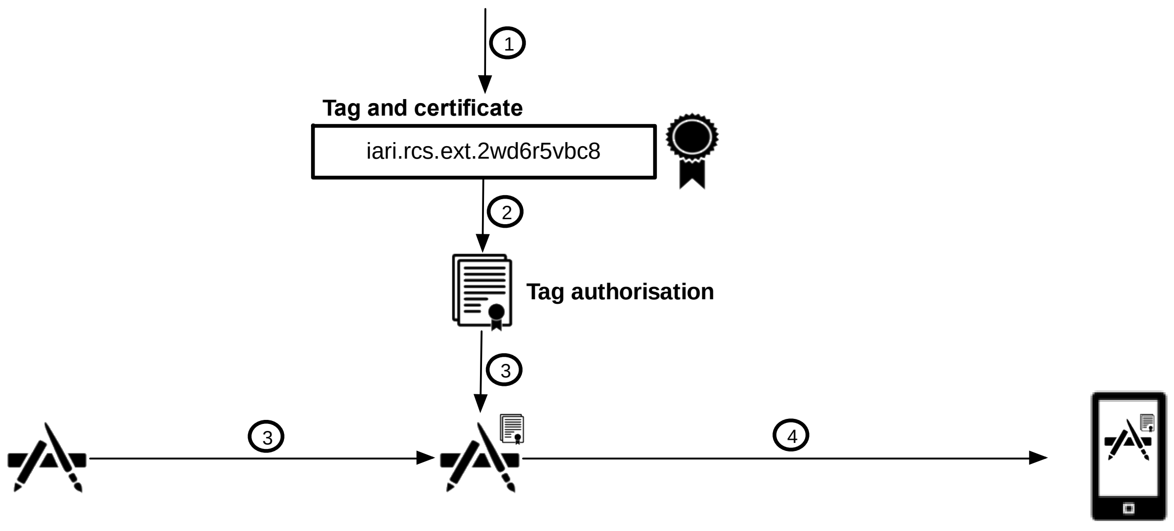


Figure 3: Third-party applications

1. The tag owner generates the unique tag string and also a tag certificate
2. The tag owner creates a signed Tag Authorisation associating that tag with that app.

3. The Tag Authorisation is packaged as part of the app by the application developer or publisher.
4. Upon installation, the RCS stack verifies that the app is authorised to use that tag.

As with the second-party case, the validated tag for an application is known by the stack and is used as the IARI in the headers of SIP traffic generated by the application. This can be used for application discovery, for the establishment of private app-to-app virtual communications, and also to mediate access to the service, or to specific sensitive APIs, depending on the access control policy of the service provider.

The details of the processes are detailed in Section 5 below.

3.5 Application identification at protocol level

Individual RCS services operating over SIP are identified by a combination of an IMS Communication Service Identifier (ICSI) which indicates what kind of traffic this is e.g. chat, and an IARI, which identifies the application generating the traffic. The IARI shall appear in the Contact header OR in both the Contact and Accept-Contact headers. These identifiers enable the network to handle sessions appropriately and ultimately route the traffic to the correct application.

The IARI for an app shall be indicated in the Contact header for all sessions initiated by an app. This enables the network to identify any such session as being associated with that app.

Traffic from an app shall either be routed to

- the native app for that traffic type, (e.g. native messaging inbox) known as app-to - native messaging, [see 2.4, use case 3]
  or
- to the same app (i.e. an app with the same IARI feature tag) known as app-to-app traffic [see 2.4 use case 2]

Where app-to-native routing is required the IARI is indicated in the Contact header only, indicating that the session should be handled by the default application for that kind of traffic as indicated by the ICSI.

Where app-to-app routing is required the IARI is also indicated in the Accept-Contact header, signifying that the session should not be handled by the default application for the indicated ICSI, and should instead be handled by the app on the receiving device with the same IARI. If no app with a matching IARI is present on the receiving device no session is established.

4 Terminal API access control

4.1 Introduction

This section sets out the proposed approach to access control for the terminal APIs, including the mechanisms proposed for independent authorisation of apps and their access attempts for “controlled openness”.

4.2 Scenarios

The following API access scenarios are addressed in this access control proposal.

Access to APIs for core services by "native" apps integrated at manufacture. The OEM will integrate the RCS stack and conforming API implementation. The natively integrated apps – whether for core services or for any MNO-specific or other RCS extensions – are required to be able to access the terminal API.

Access to APIs for core services by downloaded apps authored or approved by a MNO or a developer working on their behalf. This use-case applies to those apps offered by an MNO as a replacement native app giving access to one or more core services, possibly as part of a broader MNO service offering. These apps are preferably authorised once and distributed in a way that allows them to run on each manufacturer's products without per-device modification or configuration.

Access to APIs for core services by downloaded apps authored by Independent ISVs. This use-case applies to those apps that are simply intended to provide a different UX for the same service, or to provide a unified UX for RCS core services in parallel with third-party services (e.g. as with SMS and Google Hangouts unified in the Hangouts app).

NOTE: It is not intended to enable access by third-parties to core services initially, but the intention is that this proposal addresses this use-case and that the required enabling mechanisms are provided for in the RCS specification and the RCS stack.

Access to APIs for RCS extensions by downloaded apps (both second-party and third-party). This use-case applies to apps using RCS services to establish private app-to-app communication using a custom IARI.

4.3 API sensitivity

APIs have differing levels of sensitivity. Certain RCS APIs, whilst having a potential privacy implication for the user, do not expose the ability to generate significant traffic, and it is acceptable to permit access based only on user approval. The service discovery APIs would belong to this category. These are categorised as **open** APIs.

Other APIs expose the ability to initiate and consume sessions for various core services, and it is argued that these are open to abuse if they are made freely available to third-party apps. For various reasons – such as the need to prevent systematic violations of terms of service, or to prevent the unsustainable demands being placed on the RCS stack and service – it is proposed that most RCS communication APIs are accessible in a controlled way and not solely subject to user approval. These are categorised as **controlled** APIs.

4.3.1 Open API access

APIs categorised as open are either freely usable without permission, or are associated with a user-controlled permission; in Android these are permissions with "normal" or "dangerous" protection level. The user is prompted at the time the application is installed, and can later review these permissions via the settings application. The permissions associated with each specific API are listed in the RCS terminal API specification.

The RCS stack declares these permissions, and checks that a calling application holds the necessary permissions when making an API call.

The proposal is that of the RCS Terminal APIs only the Capability Discovery API shall be Open.

4.3.2 Controlled API Access

APIs categorised as controlled are intended to be open, and callable by third-party applications, but are sufficiently open to abuse that it must be possible for the MNO to withhold or withdraw access. These APIs will typically also have privacy or other sensitivities for the user, and therefore require install-time approval in the same way as for permissions at the dangerous protection level.

Access to controlled APIs may be additionally mediated by the MNO, based on a validated IARI; this is a real time access check, made by the network application gateway in the same way as if a network API call was being made with application-specific credentials. Since the check is online, and the check is made against a request validation endpoint exposed by the MNO, the MNO has ultimate control over whether or not any given access request is permitted.

NOTE: The approach described is dependent on the successful deployment of the infrastructure for developer and app registration for RCS network APIs, the operators' integration with the OneAPI Exchange for federation of authentication, and the operators' implementation of their own authentication and access control functionality for network APIs.

All RCS Terminal APIs other than Capability Discovery shall be controlled APIs.

5 Second-party tag management

5.1 Introduction

This section outlines the mechanisms for the management of second-party extensions that use delegated ranges of the tag space.

5.2 Overview

Each delegated tag range has an owner – this might be the MNO or an entity working on the MNO's behalf. Any given extension is permitted to use a tag within that range if the owner has authorised the application's author (who, in the Android application model, is also the signer of the app).

The owner of a tag range creates a certificate, a **IARI range certificate**, which is ultimately used by the RCS stack to determine whether or not a given app is authorised to use an IARI within that range.

The RCS stack must be configured with the certificates for the delegated tag ranges that it supports. This configuration, the tag ranges, certificates and any other range-related metadata is provided to the device along with other provisioning data. Note that any ongoing update to the set of certificates would only be necessary to react to expiring or invalidated certificates, or to incorporate new tag ranges; new configuration is not required for new tags or applications individually within a range already covered by certificates. The format for the relevant component of the provisioning data is defined in Section 8.

Inorder to publish an app that uses a tag within that range, a developer must be granted authorisation by the tag range owner; this is captured in a signed **IARI Authorization** document that references the developer's release certificate and is signed by the private key of the tag range owner. The developer then writes their app in the normal way, includes the IARI Authorization document in the application package, and uses his normal developer private key to sign and then publish the app. Note that the tag range owner does not issue any developer keys and does not sign apps but they are able to control, independently, which apps (or, strictly, which application signers) use which tags.

The RCS stack enforces the access restriction when the app attempts an operation that depends on that tag; the IARI Authorization document is checked against the app, and the signature in the IARI Authorization document is checked against the known tag range certificates held by the stack. For all operations, including those that do not use the application's tag directly, the tag is presented as the IARI in the SIP session, so all traffic generated by a second-party app is explicitly attributable to that app.

Revocation of second-party tag authorisations, by revocation of the end-entity certificate in the tag authorisation signature is not proposed at this stage, although it would be possible if the owner of the tag range operated a Certificate Authority (CA) and associated Certificate Revocation List (CRL) or Online Certificate Status Protocol (OCSP) endpoints. The format for the signature included in an IARI Authorization document also provides for embedded CRL data. Revocation though may be accomplished by invalidation of the IARI using the OneAPI Exchange application blocking function.

The steps are summarised in the figure below, with the individual steps explained in greater detail in the following sections.

The detailed format and processing requirements for an IARI Authorization document are provided in Section 7.

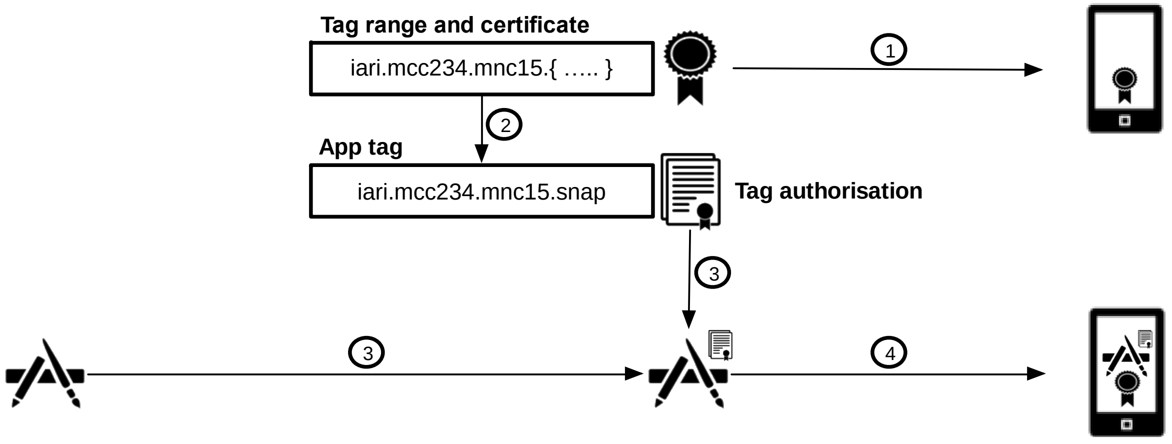


Figure 4: Second-party tag management

1. Tag range owner releases the tag range certificate, which is configured with the RCS stack on each device that supports that range.

2. The tag range owner creates a specific tag for an app, and creates a signed IARI Authorization associating that tag with that app.
3. The IARI Authorization is packaged as part of the app by the application developer or publisher.
4. Upon installation or first use, the RCS stack verifies that the app is authorised to use that tag.

Tag range owner issues Tag range certificate

The tag range certificate is a certificate that identifies the tag range as a Subject Alternative Name (SAN) extension of type URI. The range is represented as a “range expression” a simple glob expression, itself a URI, that matches all IARIs that belong to the range. The certificate could be self-signed by the tag range owner, or could in future be issued by a CA if suitable arrangements were put in place.

The tag range owner arranges for the relevant RCS stacks to embed that certificate. This could be static and may also be updatable by remote configuration (see below).

5.3 Tag range owner issues tag

When a tag is required for a specific service, the tag range owner issues the tag according to its own naming rules. There are no general restrictions on the format of a name, and would be expected that the tag range owner would follow formatting conventions (a hierarchy of tag names with dot-separated path elements) and reasonably descriptive identifiers.

5.4 Application developer creates Developer Keys

An application developer who wishes to publish an application must create his “release key” which in fact consists of a private key, the **Developer private key** and a corresponding **Developer Certificate** embedding the **Developer public key**. These are created locally by a developer using the standard Android toolchain.

This step applies both to second-party and third-party extensions, and is routine for any application developer of Android applications for public distribution.

The process is illustrated in the figure below.

App developer creates release keys

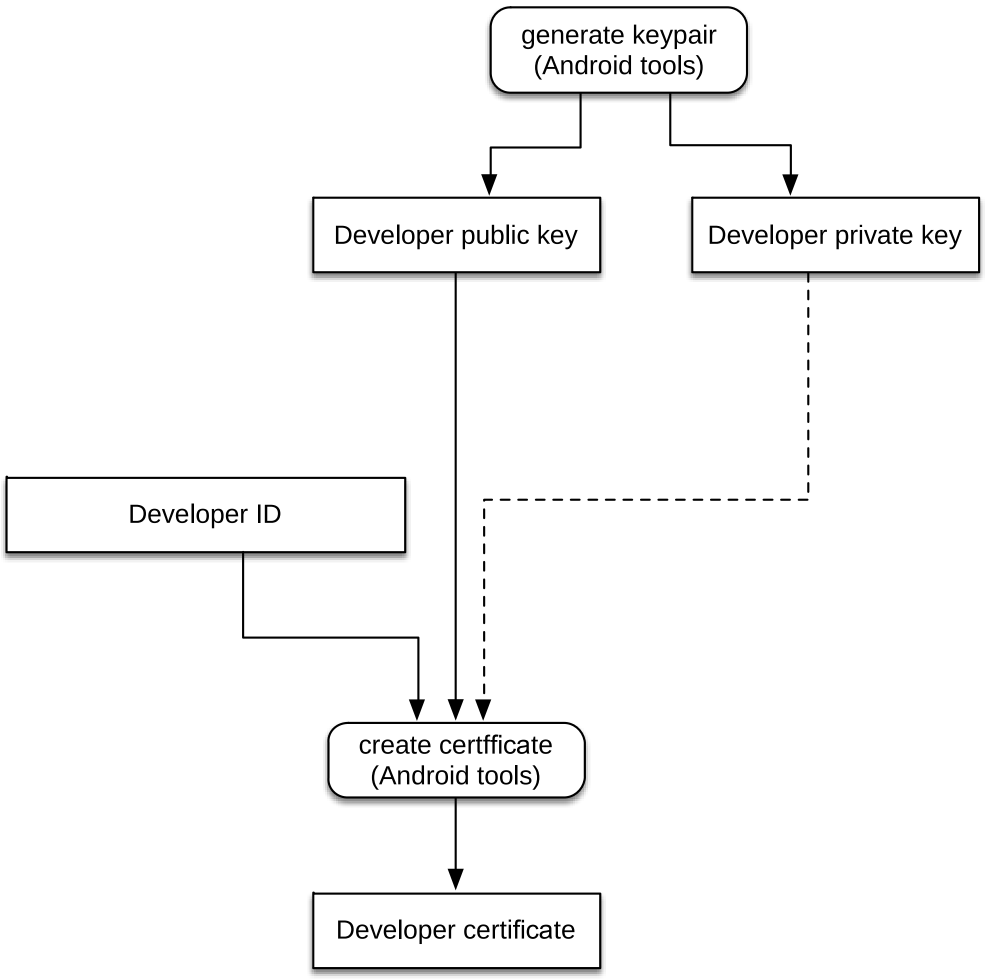


Figure 5: App developer creates release keys

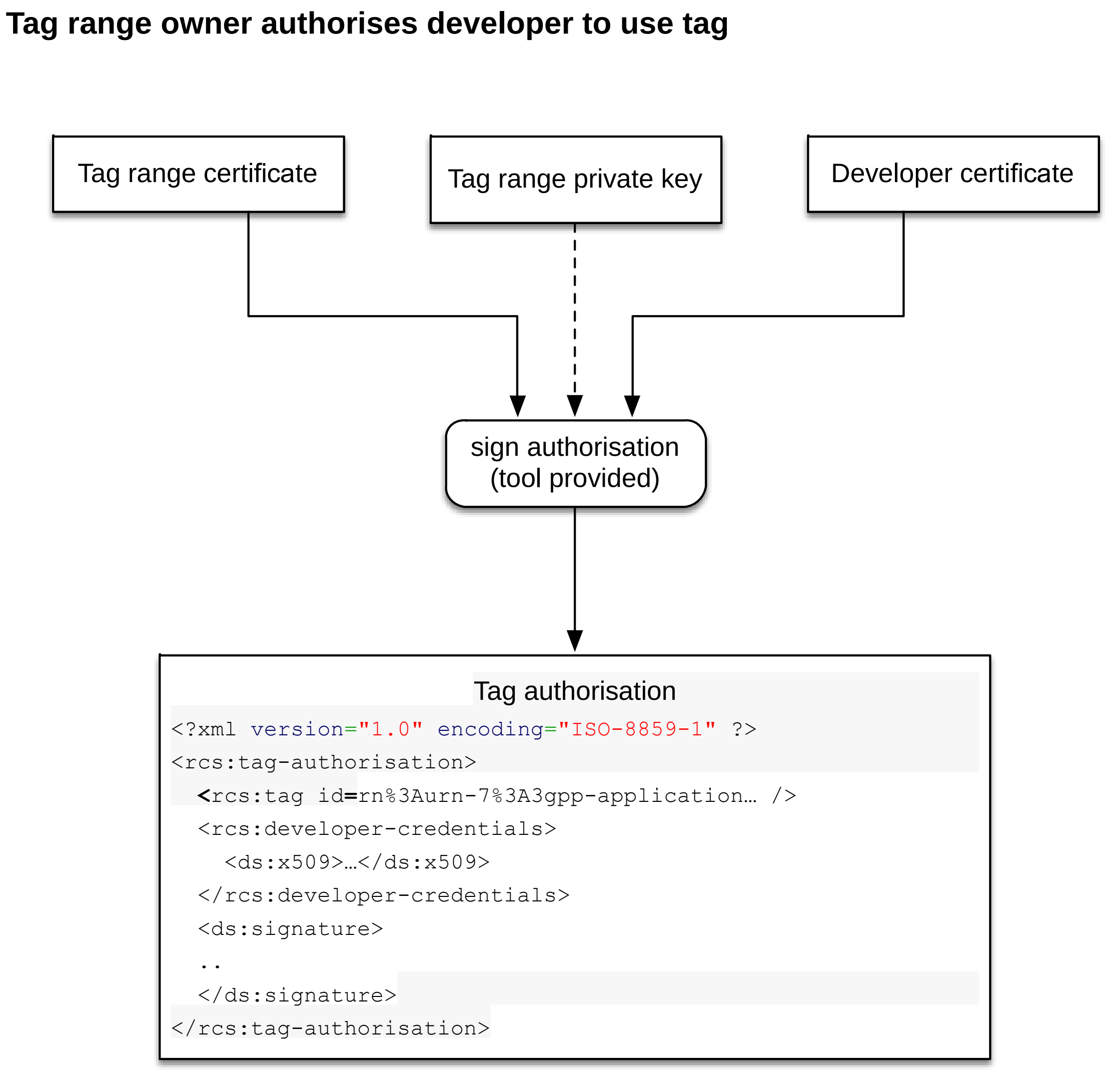
5.5 Tag owner authorizes developer to use tag

This is the step in which the owner of a Tag range certifies that a specific developer is granted authority to use tag issued from that range. The authority is represented by creating a signed document in which the Tag and the Developer Certificate are signed by the Tag range private key. Only the Tag range owner can do this, but anyone with access to the Tag range certificate can verify the authenticity of such a document. This document is referred to as the **IARI Authorization**.

The IARI Authorization is proposed to be an XML document in an GSMA-owned namespace that contains an **detached XML Digital Signature** that references the specific Tag range, Tag, the application package certificate and optionally the package ID. The signature itself is made using the Tag range private key and the signature embeds any intermediate certificates required other than the Tag range root certificate itself. The detail of the IARI Authorization document format is provided in Section 7.

This step must be performed by the Tag range owner, and he can use RCS-provided tools. Once the IARI Authorization document exists, it can be passed to the Developer and may then be used to sign multiple versions of the application package (or even multiple application packages) without further reference to the Tag range owner.

The steps for creation of the IARI Authorization document are shown in the figure below.


Tag range owner authorises developer to use tag
Tag range certificate
Tag range private key
Developer certificate
sign authorisation (tool provided)
Tag authorisation

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rcs:tag-authorisation>
  <rcs:tag id=rn%3Aurn-7%3A3gpp-application… />
  <rcs:developer-credentials>
    <ds:x509>…</ds:x509>
  </rcs:developer-credentials>
  <ds:signature>
  ..
  </ds:signature>
</rcs:tag-authorisation>
```

Figure 6: Ta range owner authorizes developer to use tag

5.6 Application developer creates and releases an app using the tag

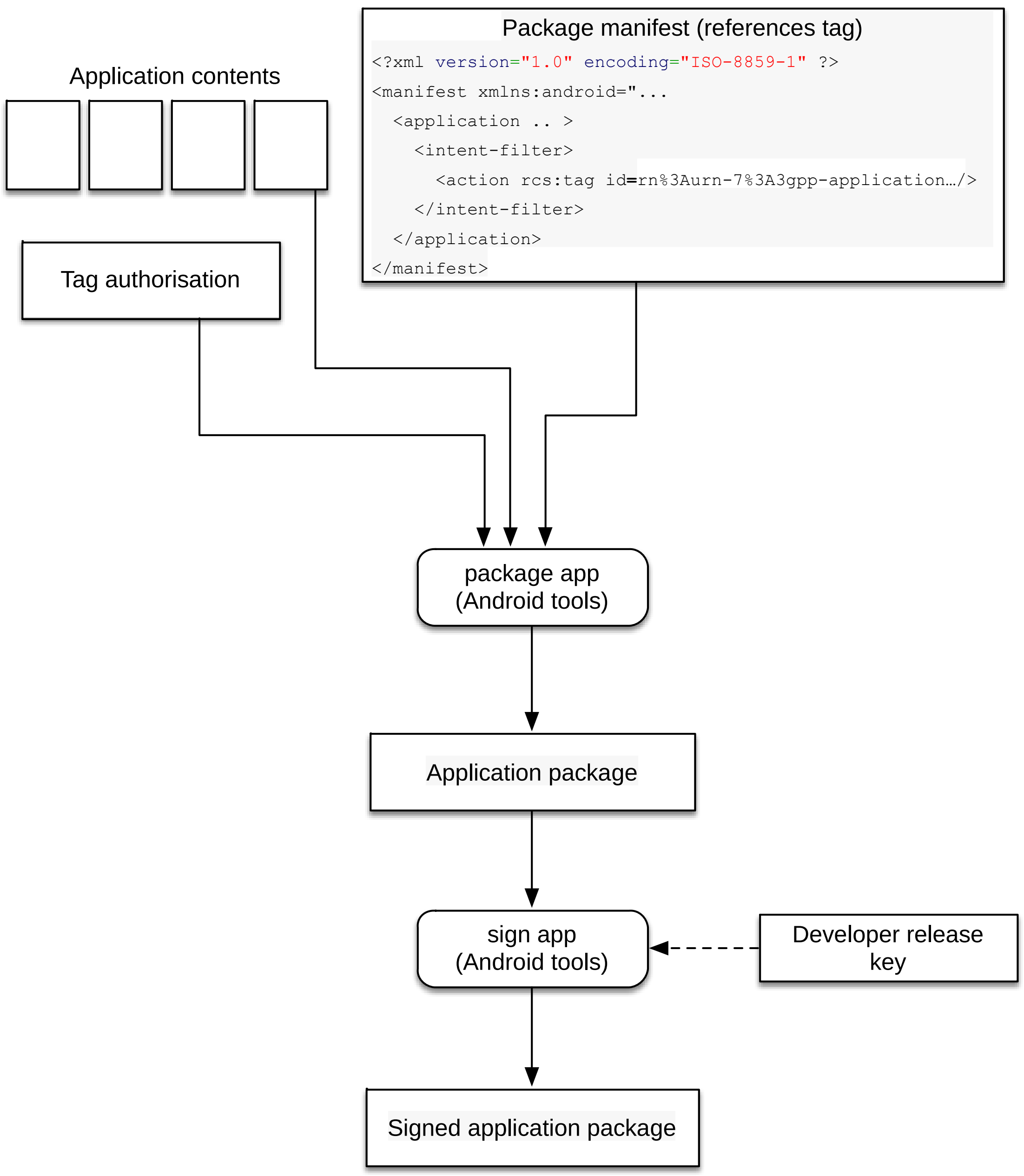
By including the IARI Authorization in the application package, the developer has bound the app to that tag, and the RCS stack can ensure that IARI is visible in sessions originated or terminated by that app.

In addition, if the application is using the tag for private app-to-app sessions, it will typically need to reference the tag string in its code and/or its manifest. For example, the GSMA MultimediaSession API requires there to be a registered intent filter whose action is the tag, in order that the application can be woken to handle incoming sessions for that tag. The application developer must develop the app and reference the tag in the manner required in the spec.

Now the application developer must add the IARI Authorization to his application package. In the case of an Android application package, the IARI Authorization is included as a standalone XML resource document (i.e. under res/xml/), with the resource ID being referenced by a well-known meta-data element in the manifest. In principle, multiple IARI Authorizations may exist within a single application package.

Once the IARI Authorization document is added, the developer signs and releases his app in the usual way. The steps are illustrated in the figure below.

App developer creates and releases app using tag



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<manifest xmlns:android="...
  <application .. >
    <intent-filter>
      <action rcs:tag id=rn%3Aurn-7%3A3gpp-application…/>
    </intent-filter>
  </application>
</manifest>
```

Figure 7: App developer crates and release app using tag

5.7 Stack validates application

Once the app has been downloaded and installed by a user, the stack can perform certain processing to validate the RCS-related declarations made by the app. This might apply at installation time, or on the first occasion that the application makes a relevant call to the RCS stack. The result of this processing might be that the application is not valid, in which case it would not be permitted to use the RCS API, or that the application is valid, and information is generated that will later be used at runtime to validate stack operations performed by the app.

Validation of the app by the stack consists of verifying each of the constituent elements of the tag-related information. These are as follows.

1. **Validate app tag usage.** The stack must verify that an IARI Authorization is present for each tag declared in the manifest. Each of the following steps must be performed for each Tag if there is more than one present.
2. **Verify package signature and ID correspond to IARI Authorization.** For each Tag authorisation present, the stack must verify that the authorisation relates to the signer of the present package and that the package ID matches the given ID if present. This is done by comparing the Developer certificate information in the Tag authorisation with the package signer certificate available from the package manager.
3. **Validate IARI Authorization.** This step consists of the validation of the well-formedness of the IARI Authorization document and validation of the signature. Together, these checks confirm that the document was issued and signed for the app in question, and those details or the signature have not since been modified.
4. **Verify IARI Authorization signature**. This establishes that the signature is not only valid, but that it was created by a party that is trusted (either directly, or by the chain of trust in the certificate path). This relies on the IARI range certificates configured with the stack.
5. **Associate application package with tag.** If all of the steps above prove that the package and IARI authorization are valid, the package ID is associated with the Tag(s). This association is later used at runtime.

If any of the steps above fail, the stack does not need to remove the app but it must decline any RCS operations attempted using the stack.

The steps are illustrated below.

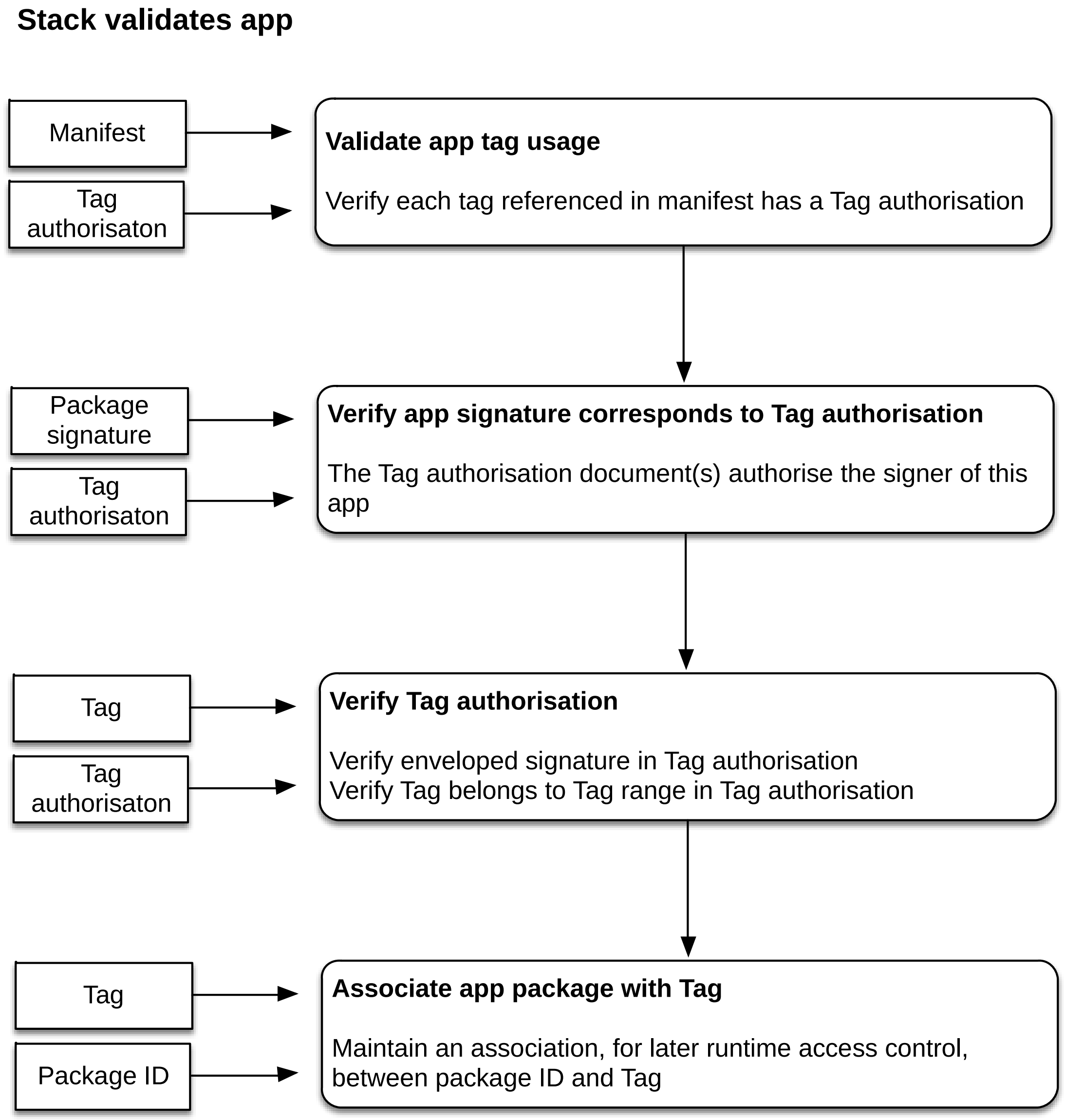

Stack validates app
Manifest
Tag authorisaton
Validate app tag usage
Verify each tag referenced in manifest has a Tag authorisation
Package signature
Tag authorisaton
Verify app signature corresponds to Tag authorisation
The Tag authorisation document(s) authorise the signer of this app
Tag
Tag authorisaton
Verify Tag authorisation
Verify enveloped signature in Tag authorisation
Verify Tag belongs to Tag range in Tag authorisation
Tag
Package ID
Associate app package with Tag
Maintain an association, for later runtime access control, between package ID and Tag

Figure 8: Stack validates app

5.8 Stack validates runtime invocation

Each time an application invokes an operation on the stack that depends on the tag, the stack must confirm that the calling package is authorized. This is performed by:

- **obtain the package ID of the caller.** Using the Binder, the stack can obtain the package identity of the caller of any AIDL methods.
- **Verify the package ID has been associated with the tag.** This confirms the authorisation to use the tag based on the previously determined association.

This is illustrated below.

Stack validates runtime invocation

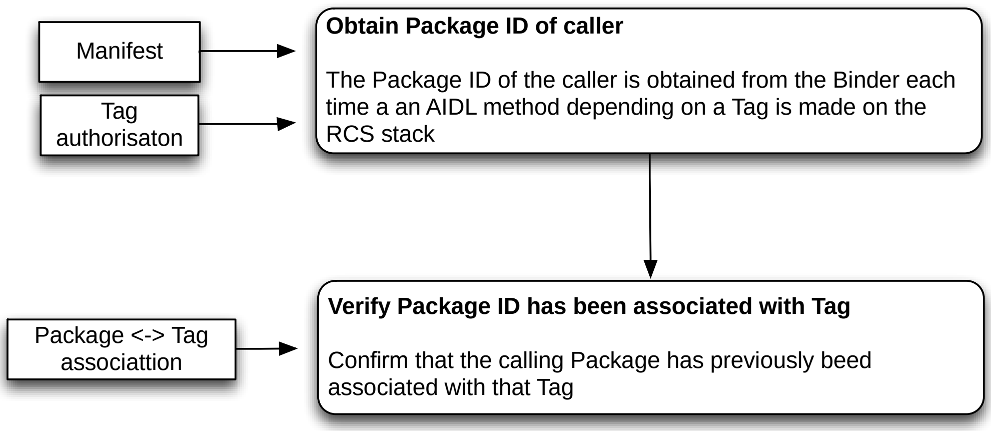

Manifest
Tag authorisaton
Obtain Package ID of caller
The Package ID of the caller is obtained from the Binder each time a an AIDL method depending on a Tag is made on the RCS stack
Package <-> Tag associattion
Verify Package ID has been associated with Tag
Confirm that the calling Package has previously beed associated with that Tag

Figure 9: Stack validates runtime invocation

6 Third-party tag management

6.1 Introduction

This section outlines the mechanisms whereby third parties can create and use tags without a requirement for a central issuer. Much of the process is common with the second-party case described above.

6.2 Overview

Unlike the second-party case, there is no pre-assigned tag range that is available exclusively for the use of a single third-party entity. Instead, for the third-party case there is a single "ext" tag range for tags, and the tag generation process ensures that unique tag strings are generated.

Instead of there being a single certificate for a tag range, each individual tag is associated with a keypair generated independently by a tag owner. The tag string itself is derived from the public key of that keypair. The owner keeps the private key private, but uses it to prove that they own the tag, and subsequently to prove that they have authorised particular apps to use the tag.

In order to publish an app that uses such a tag, a developer must be granted authorisation by the tag owner; in the same way as for second-party apps, this is captured in a signed IARI Authorization document that references the developer's release certificate and is now signed by the tag private key. The developer writes their app in the normal way, includes the IARI Authorization document in the application package, and uses their normal developer private key to sign and then publish the app.

This means that it is possible for the stack – or anyone else - to verify that the app was authorised by the owner to use that tag, without there ever having been any central authority either to issue the tag, or to issue the certificate used to sign the app. The integrity of the tag depends only on how carefully the owner manages his own tag private key, and the cost of creating and managing keys falls to the developer, in just the same way as happens already for his developer private key.

Neither the tag owner or developer is identified by the signature; the chain of tag, IARI Authorization and package signature only assures that the developer was authorised by the owner of the tag string to use that tag.

A separate issue from that of authority to use a tag is that of legitimacy of an application or tag. It is possible that an application uses a custom tag to abuse the RCS service and that there is a need to disable that app or its interaction with the service. Since there is no central issuer, it is not possible simply to revoke the Tag certificate if a specific app or tag is judged to be malware. Instead, a combination of measures, including blocking of specific tags in the network, together with restrictions enforced by the stack (applying a blacklist of tags) is proposed.

The steps are summarised in the figure below, with the individual steps explained in greater detail in the following sections.

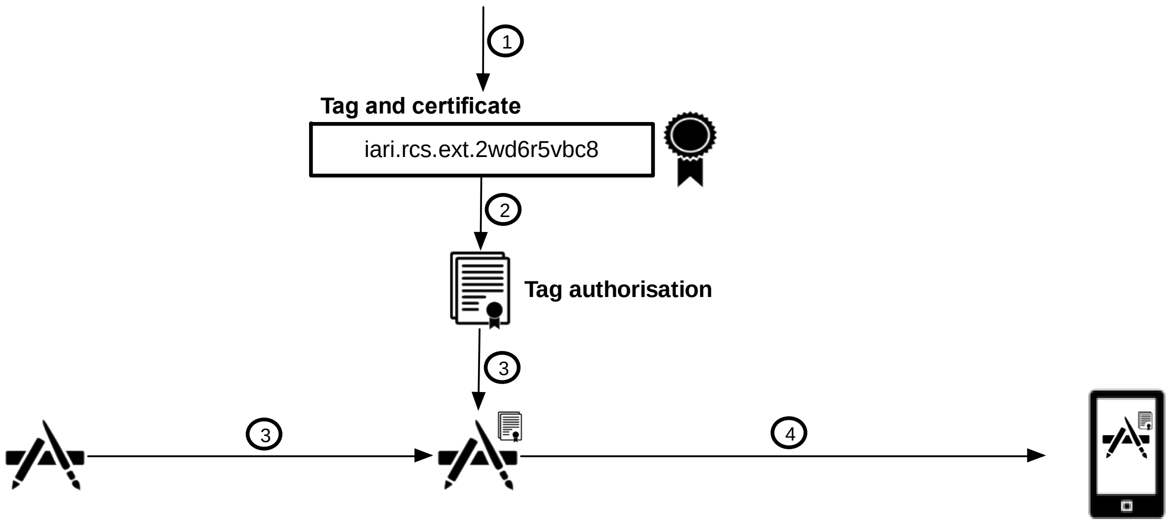


Figure 10：Third-party tag management

1. The tag owner generates the unique tag string and also a tag certificate
2. The tag owner creates a signed IARI Authorization associating that tag with that app.
3. The IARI Authorization is packaged as part of the app by the application developer or publisher.
4. Upon installation, the RCS stack verifies that the app is authorised to use that tag.

6.3 Tag owner creates tag

A public/private keypair is generated locally by the tag owner as the basis for the tag; tools are provided for the tag owner to do this. The private key – referred to in the following as the **Tag private key** - is kept secret by the tag owner, and used only to sign Tag authorisation documents (see below).

The tag string itself is formed by hashing the corresponding public key – the **Tag public key** - and prepending the custom tag prefix:

```
urn:urn-7:3gpp-application.ims.iari.rcs.ext.ss<hashed tag public key>
```

The hash function is to be decided; after hashing, and encoding with an encoding suitable for inclusion in a tag urn string, the hash length should preferably not exceed 40 bytes. An SHA-224 hash, encoded with URL-safe Base64 (RFC 4648), which has an encoded length of 38 bytes, is suggested.

The tag owner creates a self-signed certificate using the tag keypair and containing the tag string as a Subject Alternative Name (SAN) entry. This certificate will be used in any signature based on the tag.

The steps are illustrated below.

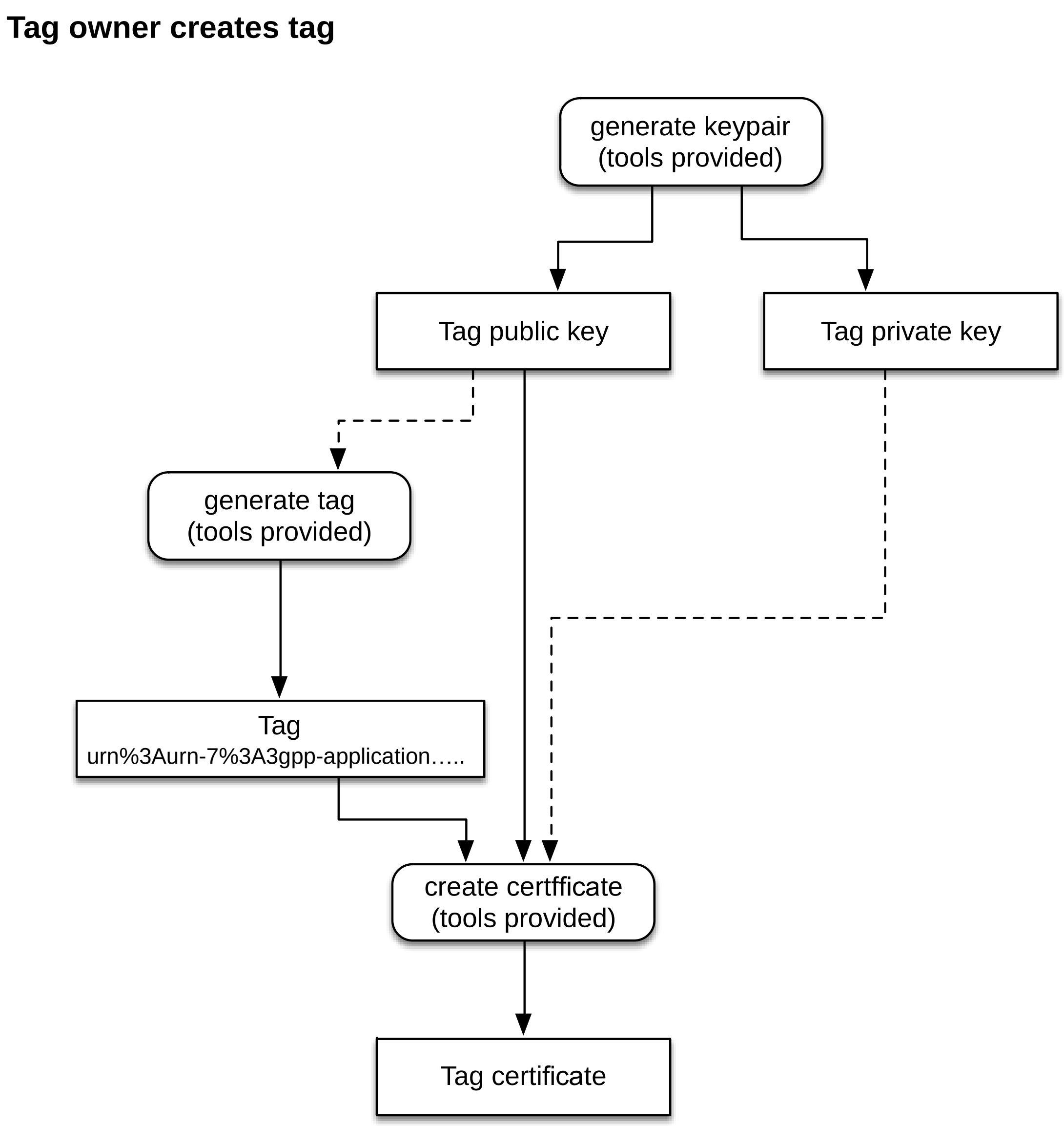

Tag owner creates tag
generate keypair (tools provided)
Tag public key
Tag private key
generate tag (tools provided)
Tag urn%3Aurn-7%3A3gpp-application…..
create certfficate (tools provided)
Tag certificate

Figure 11: Tag owner creates tag

6.4 Application developer creates Developer Keys

This is the routine step for a developer of creating “release keys” to be used for application signing for public distribution.

An application developer who wishes to publish an application must create their “release key” which in fact consists of a private key, the **Developer private key** and a corresponding **Developer Certificate** embedding the **Developer public key**. These are created locally by a developer using the standard Android toolchain.

The process is the same as for the second-party case, and is illustrated in section 5.5 above.

6.5 Tag owner authorizes developer to use tag

This is the step in which the owner of a tag decides that a specific developer is granted authority to use the tag. Typically the tag owner and the developer would be the same entity, but the process does not depend on this.

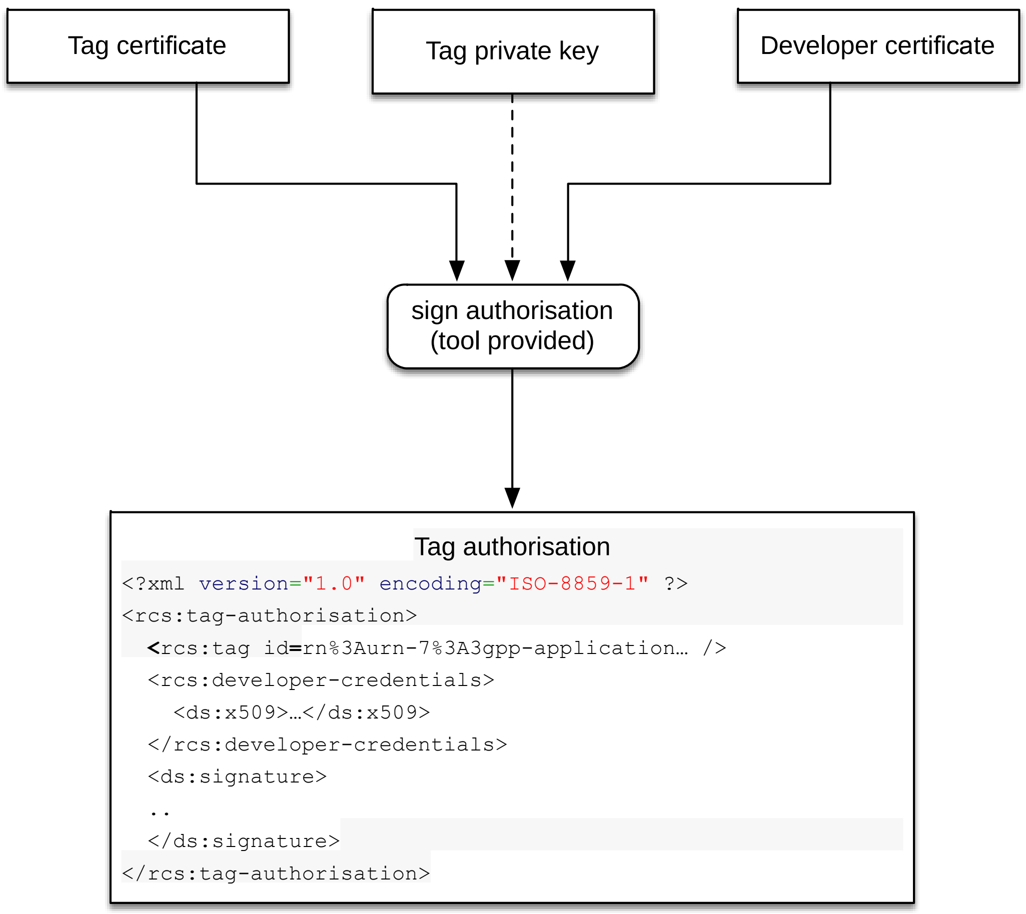
Just as in the second-party case, the authority is represented by creating a signed document in which the tag and the Developer Certificate are referenced, this time by the tag private key. Only the tag owner can do this, but anyone with access to the Tag certificate can verify the authenticity of such a document. This document is again referred to as the **IARI Authorization**.

The IARI Authorization is an XML document in an GSMA-owned namespace that contains an **detached XML Digital Signature**. The IARI Authorization document that references the specific tag, the application package certificate and optionally the package ID. The signature itself is made using the tag private key and the signature embeds the Tag Certificate.

This step must be performed by the tag owner, and they can use RCS-provided tools. Once the tag authorisation document exists, it can be passed to the developer and may then be used to sign multiple application packages without further reference to the tag owner.

The steps for creation of the Tag authorisation document are shown in the figure below.

Tag owner authorises developer to use tag



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rcs:tag-authorisation>
  <rcs:tag id=rn%3Aurn-7%3A3gpp-application… />
  <rcs:developer-credentials>
    <ds:x509>…</ds:x509>
  </rcs:developer-credentials>
  <ds:signature>
  ..
  </ds:signature>
</rcs:tag-authorisation>
```

Figure 12: Tag owner authorises developer to use tag

6.6 Application developer creates and releases an app using the tag

This step is the same as for the second-party case, described in Section 5.7 above.

An application developed to make use of the tag will typically need to reference the tag string in its manifest. For example, the GSMA MultimediaSession API requires there to be a registered intent filter whose action is the tag, in order that the application can be woken to handle incoming sessions for that tag. The application developer must develop the app and reference the tag in the manner required in the spec.

The application developer must then add the IARI Authorization to their application package. In the case of an Android application package, the IARI Authorization is included as a standalone XML resource document (i.e. under res/xml/), with the resource ID being referenced by a well-known meta-data element in the manifest. In principle, multiple IARI Authorizations may exist within a single application package.

Once the IARI Authorization document is added, the developer signs and releases his app in the usual way. The steps are illustrated in the figure in Section 5.7 above.

6.7 Stack validates application

This step is the same as for the second-party case, described in Section 5.8 above.

Once the app has been downloaded and installed by a user, the stack can perform certain processing to validate the RCS-related declarations made by the app. This might apply at installation time, or on the first occasion that the application makes a relevant call to the RCS stack. The result of this processing might be that the application is not valid, in which case it might not be permitted to use the RCS API, or that the application is valid, and information is generated that will later be used at runtime to validate stack operations performed by the app.

Validation of the app by the stack consists of verifying each of the constituent elements of the tag-related information. These are as follows.

1. **Validate app tag usage.** The stack must verify that a Tag authorisation is present for each tag declared in the manifest. Each of the following steps must be performed for each tag if there is more than one present.
2. **Verify app signature corresponds to Tag authorisation.** For each Tag authorisation present, the stack must verify that the authorisation relates to the signer of the present package. This is done by comparing the Developer certificate information in the Tag authorisation with the package signer information available from the package manager.
3. **Validate IARI Authorization.** This step consists of the validation of the well-formedness of the IARI Authorization document and validation of the signature. Together, these checks confirm that the document was issued and signed for the app in question, and those details or the signature have not since been modified. The validation also establishes that the tag string matches the public key of the Tag certificate.
4. **Associate application package with tag.** If all of the steps above prove that the package and authorisation are valid, the package ID is associated with the Tag(s). This association is later used at runtime.

If any of the steps above fail, the stack does not need to remove the app but it must decline any RCS operations attempted using the stack.

The steps are illustrated in the figure in Section 5.8 above.

6.8 Stack validates runtime invocation

Each time an application invokes an operation on the stack that depends on the tag, the stack must confirm that the calling package is authorized. This is performed by:

- **obtaining the package ID of the caller.** Using the Binder, the stack can obtain the package identity of the caller of any AIDL methods.
- **Verifying the package ID that has been associated with the tag.** This confirms the authorisation to use the tag based on the previously determined association.

This is illustrated below.

Stack validates runtime invocation

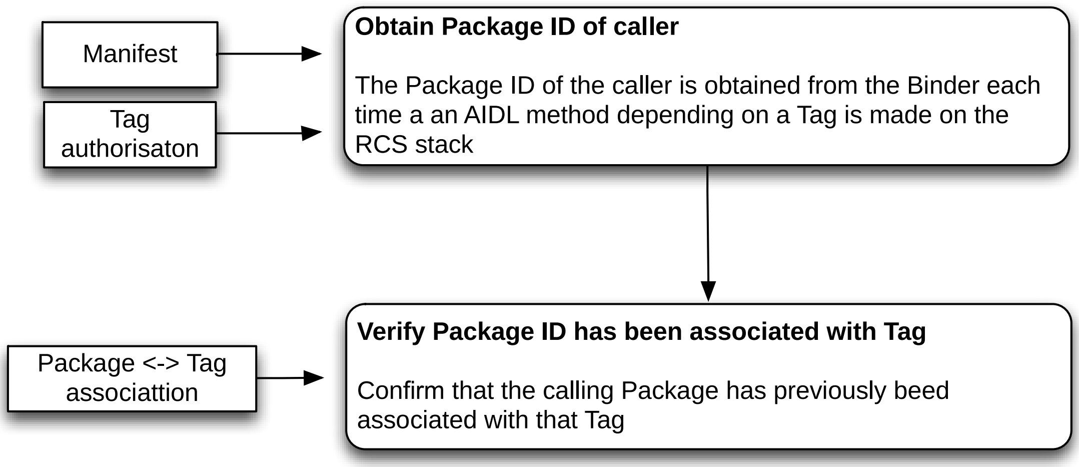

Manifest
Tag authorisaton
Obtain Package ID of caller
The Package ID of the caller is obtained from the Binder each time a an AIDL method depending on a Tag is made on the RCS stack
Package <-> Tag associattion
Verify Package ID has been associated with Tag
Confirm that the calling Package has previously beed associated with that Tag

Figure 13: Stack validates runtime invocation

6.9 Tag validity: assurances provided to the user and developer

A valid stack will ensure that only an authorized app can listen for, and initiate, sessions for a given tag. Non-conforming stacks, or stacks running on rooted devices, cannot provide those assurances; therefore we have to understand what we can and cannot assure when such configurations are inevitably present on the network.

A user that ensures their device is not rooted, and runs a valid stack, will know that:

- malware cannot masquerade as validly supporting a tag, and will therefore not advertise that capability to other devices;
- such malware cannot be invoked in response to inbound sessions for a tag that it does not own;
- such malware cannot intercept the messages sent and received between valid peers on a tag it does not own.

Such a user cannot know that inbound sessions are not from malware – i.e. another party might have a non-confirming stack or rooted device that is running malware that masquerades as being authorised to use a tag. Such malware could initiate a session to a valid app. Developers of apps using custom tags must therefore be aware of the risk of unauthorised inbound sessions and protect themselves appropriately. In this sense, enabling inbound sessions on a tag is similar to opening a port on an internet-connected device; when there is a connection to that port, the listening application must implement measures corresponding with its own level of risk to establish the validity of those sessions.

Similarly, a valid app discovering a remote peer and initiating a session to that peer cannot know that it is a legitimate app running on a valid stack. There is no means at the RCS level to authenticate the app itself if it might be on a compromised device or stack. Developers of apps using custom tags must therefore be aware of the risk of unauthorised peers and protect themselves appropriately.

6.10 Tag registration and developer identification

The mechanisms described allow application developers to create custom tags and authorise apps to use those tags. These processes are decentralised; there does not need to be a central issuing authority to create tag strings or manage the allocation to avoid conflicts, or issue certificates for tags.

However, it is expected that there will be nonetheless business requirements relating to developer identification and there may be a requirement for developers to enter into terms of service or commercial license agreements. This means that there would be some process that gates the creation of tags. In practice this could be in the form of a web-based portal that tag owners use to register their details plus metadata for the tag – the tag string itself, the purpose of the tag, certificate details, status, and perhaps also a list of all apps or developers that have been authorised to use it.

Based on this information, it would also be possible to expose that tag registry in searchable form. Such a database would not necessarily be a runtime dependency of the RCS stack or the service, but may be useful in tracking apps and their distribution. A registry could include information for apps or tags that are blocked, or whose status is under review.

A possible future requirement is for the registry to link to technical specifications and interoperability tests for tags associated with services that are intended to support independent implementations.

6.11 Tag pre-registration

A service provider might have a requirement for more explicit control over third-party apps, requiring explicit approval before granting any access to controlled RCS terminal APIs. For example there might be a requirement for positive identification of the developer, or requirement that the developer is bound by a service agreement, before access is granted.

It is possible to implement a check in the network based on the IARI presented in a session when an application creates a session. It is possible for such an online check to be performed by the RCS application gateway if the MNO provides a request validation API (similar to or actually using the OneAPI Exchange request validator). Since the check is online, and the check is made transparent against a request validation endpoint used by the MNO, the provider has ultimate control over whether or not any given access request is permitted.

A system of request validation is defined as part of the OneAPI Exchange support for the RCS network API. This envisages application developers registering with a single network operator, the “Developer Operator” (DO). Users of that application who are subscribed to a different “Serving Operator” (SO) are able to access the service, as the Application Gateway at the SO can make an API call to a Request Validator API in the OneAPI exchange, which acts as a referrer for validation requests fulfilled by the DO’s infrastructure. This Request Validator API also verifies the developer is signed up the Serving Operator Terms & Conditions, the application has been approved (manually or automatically) by the Serving Operator and neither the Serving or Developer Operators have currently blocked the application.

NOTE: The approach described is dependent on the successful deployment of the infrastructure for developer and app registration for RCS network APIs, the operators' integration with the OneAPI Exchange for federation of authentication, and the operators' implementation of their own authentication and access control functionality for network APIs.

6.12 Tag blocking

Even with tag pre-registration, it is possible that an app, after having been authorised, is found to behave in a way that abuses the network. This means that it is necessary to have a way to disable access to RCS services for specific applications or tags.

Since there is no central issuing authority, it is not possible to use a revocation mechanism based on the tag or tag range certificate. However, since the tag is visible to the stack and the network, both for private app-to-app services and core services, it is possible to block access at either point.

Network blocks may be implemented using the same underlying mechanism as for Tag pre-registration. The RCS Application Gateway, with visibility of the IARI for any app, can check that IARI against a backlist of blocked apps. An MNO (as Serving Operator) may choose to block an app on that specific network, or a DO (Developer Operator) may propagate a blocked status for an app to all SOs, meaning that the blocked status would be indicated in responses to the Request Validator API.

A mechanism is provided also for a stack-enforced application blacklist. The RCS stack can enforce a restriction whereby apps belonging to a "blacklist" would be unable to perform RCS operations. The blacklist would be maintained centrally, or by each MNO, using the same underlying services as described above for the network block.

The advantage of a device-based block, in addition to the network block, is that a device can block applications that would otherwise flood the network with requests, placing unsustainable load on the application gateway and request validation infrastructure.

However, there are limitations:

- there is no assurance that the restriction would be enforced on rooted devices, or those with a non-approved RCS stack;
- a mechanism is required to be implemented in the stack to maintain the blacklist;
- as malware proliferates, the blacklist could grow significantly, and broadcast of updates could become a burden on the network and the stack.

The supported mechanism is based on an End User Confirmation Request (EUCR) system requests is supported by the RCS stack. A network can send a system request with type *urn:gsma:rcs:extension:control* to block a tag either temporarily or permanently. The stack is required to disable access to that specific tag for the specified period, for any apps installed that use that tag.

7 IARI Authorisation document specification

7.1 Introduction

This section defines the content and required processing for an IARI Authorisation document.

7.2 Authorization document types

Any given IARI Authorization document is one of two types:

1. An IARI Range authorisation, supporting the process described in Section 5. The document grants a specific package signer the right to use a specific IARI, on the authority of the owner of the enclosing IARI range;
2. A standalone IARI authorisation, supporting the process described in Section 6. The document represents the grant of the right for a package signer to use a specific IARI on the authority of the owner of that IARI.

These two document types have the same basic structure, with differences only in the required elements, and the characteristics of the signature.

7.3 Namespace

The **IARI Authorization namespace** URI for an IARI Authorization document is:

`http://gsma.com/ns/iari-authorization#`

No provision is made for an explicit version number in this specification. If a future version of this specification requires explicit versioning of the document format, a different namespace will be used.

7.4 `iari-authorization` element

The `iari-authorization` element serves as the container for the other elements of an IARI Authorization document.

Context in which this element is used	The `iari-authorization` element is the root element of the IARI Authorization document.
Occurrences	Exactly one, at the root element of the XML document.
Expected children	`iari`: one `range`: zero or one `package-name`: zero or one `package-signer`: one `Signature`: one
Attributes	None

7.5 `iari` element

The `iari` element represents the IARI string to which this IARI Authorization document applies.

Context in which this element is used	In the `iari-authorization` element.
Content model	A valid IRI matching the IRI token of the [IRI] specification and satisfying the format requirements of an IARI string.
Occurrences	Exactly one.
Expected children	None
Attributes	`Id`: optional, type ID.

7.6 `range` element

The `range` element represents the IARI range expression to which this IARI Authorization document applies.

Context in which this element is used	In the `iari-authorization` element. Valid for an IARI Range Authorization only.
Content model	A valid IRI matching the IRI token of the [IRI] specification and matching the format requirements of an IARI range expression.
Occurrences	Zero or one.
Expected children	None
Attributes	`Id`: optional, type ID.

7.7 `package-name` element

The `package-name` element represents the application package identifier to which this IARI Authorization document applies.

Context in which this element is used	In the `iari-authorization` element.
Content model	A string value, equalling the value of the `package` attribute in the `<manifest>` of an Android application.
Ocurrences	Zero or one.
Expected children	None
Attributes	`Id`: optional, type ID.

If an IARI Authorization does not include a <package-name> element, the authorisation applies to any application package whose package signer details match those specified in the document.

7.8 `package-signer` element

The `package-signer` element represents the application package signer to which this IARI Authorization document applies. A given `package-signer` value matches an application package if the entity certificate of one of the package signatures has a fingerprint matching the given `package-signer` value.

The fingerprint format used is the SHA1 digest of the DER-encoded representation of the certificate, represented as colon-delimited, uppercase hex-encoded bytes. An example fingerprint is:

```
    0D:25:2D:E7:A3:A7:C7:47:16:41:39:93:84:7F:1A:F6:EF:94:84:91
```

Context in which this element is used	In the `iari-authorization` element.
Content model	A string containing the SHA1 fingerprint of the package signature entity certificate.
Occurrences	Exactly one.
Expected children	None
Attributes	`Id`: optional, type ID.

7.9 `Signature` element

The `signature` element represents contains a digital signature, binding the other elements of the IARI Authorization document to a certificate. The signature represents the authority of the IARI or IARI range owner to the use of the IARI that is the subject of the document.

The certificate is either trusted as belonging to the IARI Range owner (in the case of an IARI Range authorization) or provably belongs to the IARI owner (in the case of a standalone IARI Authorization).

The `<Signature>` element must belong to the XML Digital Signature namespace:

```
http://www.w3.org/2000/09/xmldsig#
```

Context in which this element is used	In the `iari-authorization` element.
Content model	A detached XML Digital Signature conforming to the XML Signature Syntax and Processing Version 1.1 specification ([XMLDSIG]) and conforming to the additional requirements below.
Ocurrences	Exactly one.
Expected children	As required by ([XMLDSIG]) and conforming to the additional requirements below.
Attributes	`Id`: optional, type ID.

7.9.1 Algorithms, key lengths, and certificate formats

This specification relies on a user agent's conformance to [XMLDSIG] for support of signature algorithms, certificate formats, canonicalization algorithms, and digest methods. As this specification is a profile of [XMLDSIG], it makes a number of recommendations as to what signature algorithms should be used when signing a widget package to achieve optimum interoperability. See Signature Algorithms of [XMLDSIG] for the list of required algorithms.

The ***recommended signature algorithm*** is RSA using the `RSAwithSHA256` signature identifier: http://www.w3.org/2001/04/xmldsig-more#rsa-sha256.

The ***recommended key length*** for RSA is 2048 bits or greater.

The ***recommended digest method*** is SHA-256.

The ***recommended canonicalization algorithm*** is *Canonical XML Version 1.1 (omits comments)* as defined in [C14N11]. The identifier for the algorithm is http://www.w3.org/2006/12/xml-c14n11.

The ***recommended certificate format*** is X.509 version 3 as specified in [RFC5280].

7.9.2 KeyInfo

A `ds:Signature` element must include a `ds:KeyInfo` element in the manner described in [XMLDSIG] (see The KeyInfo Element for how to do this). The element can include CRL and/or OCSP information.

The signature must include a child `ds:X509Data` element within the `ds:KeyInfo`, as specified by the [XMLDSIG] specification, containing at least the entity certificate (as a `ds:X509Certiifcate`) plus, in the case of a IARI Range Authorization, such other certificates as are needed to construct a chain up to, but not necessarily including, the IARI Range owner’s root certificate. The `ds:X509Data` element may additionally include CRL and/or OCSP response information that, if included, are conveyed according to the [XMLDSIG] specification.

7.9.3 Signature properties

The Signature must include container elements for [Signature Properties] in accordance with the Signature Properties Placement section of [Signature Properties].

The `ds:SignatureProperties` must include a `Role` property whose URI attribute has value:

- http://gsma.com/ns/iari-authorization#role-range-owner for an IARI Range Authorization;
- http://gsma.com/ns/iari-authorization - role-standalone for a standalone IARI Authorization.

The `ds:SignatureProperties` must include an `Identifier` property in the manner specified in [Signature Properties].

The `ds:SignatureProperties` must include a `Profile` property whose URI attribute has value:

- http://gsma.com/ns/iari-authorization - profile.

The `ds:SignatureProperties` should include a `Created` property whose element body is a date/time in http://www.w3.org/TR/NOTE-datetime format, signifying the creation time of the signature.

7.9.4 References

A `Signature` element must contain a same-document reference to each of the `iari`, `range`, `package-name`, `package-signer` elements, where present, referencing each using a fragment URI reference to its ID.

A signature must contain a same-document reference to the `ds:Object` that contains the signature properties identified above.

7.10 IARI Authorization document processing

An RCS stack processes an IARI Authorization document associated with an application package in order to verify the right for that package to use the IARI in question.

Processing may occur on application installation, or on the first attempt to use the service, and on any subsequent attempt if processing results are not cached.

The steps for processing a document are defined below.

1. Parse the IARI Authorization document with an XML parser that is namespace-aware. If the document is not well-formed XML then the processor must terminate these steps and treat the IARI Authorization as invalid.
2. If the document element is not an `iari-authorization` element in the `iari-authorization` namespace then the processor must terminate these steps and treat the IARI Authorization as invalid.
3. For each child of the document element:

   a) If the element is the first encountered `iari` element, let *iari* be the text content of this element. Check the syntactic validity of *iari*. If the element is not the first `iari` element, it must be ignored.
   b) If the element is a `range` element, let *range* be the text content of this element. Check the syntactic validity of *range*. If the element is not the first `range` element, it must be ignored.
   c) If the element is a `package-name` element, let *package-name* be the text content of this element. If the element is not the first `package-name` element, it must be ignored.
   d) If the element is a `package-signer` element, let *package-signer* be the text content of this element. If the element is not the first `package-signer` element, it must be ignored.
   e) If the element is a `Signature` element in the XMl Digital Signature namespace, then process the signature according to the signature processing step below. If the element is not the first `Signature` element, it must be ignored.
   f) Any other element must be ignored.

4. If *iari* or *package-signer* have not been assigned after processing all of the elements, the processor must terminate these steps and treat the IARI Authorization as invalid.
5. If *range* has been assigned after processing all of the elements:

   a) Let *type* be *range*; otherwise let *type* be *standalone*.
   b) Check that *iari* satisfies the constraints of the range expression *range*.

6. Process the signature by the following steps:

   a) If signature is not a valid [XMLDSIG] signature, then the processor must terminate these steps and treat the IARI Authorization as invalid.
   b) Check that signature has a `ds:Reference` for each of the `iari`, `range`, `package-name` and `package-signature` elements present. If any such element exists without a reference, then the processor must terminate these steps and treat the IARI Authorization as invalid.
   c) Check that signature has a single same-document `ds:Reference` to a `ds:Object` container for the `SignatureProperties` in accordance with the Signature Properties Placement section of [Signature Properties].
   d) Optionally, if the `ds:Signature`'s key length for a given signature algorithm (e.g., RSA) is less than a stack-predefined minimum key length, then then the processor must terminate these steps and treat the IARI Authorization as invalid.
   e) Validate the `Profile` property against the profile URI in the manner specified in [Signature Properties]. If the profile property is missing or invalid, then the processor must terminate these steps and treat the IARI Authorization as invalid.
   f) Validate the `Identifier` property in the manner specified in [Signature Properties]. If the identifier property is missing or invalid, then the processor must terminate these steps and treat the IARI Authorization as invalid.
   g) If *type* is *range*, validate the `Role` property against the range role URI. If the `Role` property is missing or invalid, then the processor must terminate these steps and treat the IARI Authorization as invalid.
   h) If *type* is *standalone*, validate the `Role` property against the standalone role URI. If the `Role` property is missing or invalid, then the processor must terminate these steps and treat the IARI Authorization as invalid.
   i) Optionally, validate any other SignatureProperties supported by the processor in the manner specified in [Signature Properties].
   j) Perform reference validation and signature validation on the signature. If validation fails, then the processor must terminate these steps and treat the IARI Authorization as invalid.

7. If *type* is *range*, check that the IARI Authorization satisfies the trust requirements for the given *range*:

   a) Check that that root certificate has *range* as a Subject Alternative Name (SAN) entry of type URI.
   b) Check that the root certificate of the signature certificate chain is trusted by the processor for *range*;

8. Otherwise, if *type* is *range*, check that the IARI Authorization satisfies the trust requirements for the given *iari*:

   a) Check that that root certificate has *iari* as a Subject Alternative Name (SAN) entry of type URI.
   b) Check that *iari* matches the format for a standalone IARI, comprising the `urn:urn-7:3gpp-application.ims.iari.rcs.ext.ss` prefix followed by a Base64-encoded hash value.

   c) Check that the hash value is the SHA-224 hash of the public key of the signature’s root certificate.

9. Check that the application package associated with the IARI Authorization matches:

   a) If *package-name* is set, verify that it matches the value of the `package` attribute in the package `<manifest>`.
   b) Verify that one of the package signatures has an entity certificate whose fingerprint matches *package-signer*.

10. If the IARI Authorization is valid according to the above steps, then the stack may permit the application to use iari.

8 IARI range configuration specification

8.1 Introduction

For those MNOs wishing to enable RCS Terminal APIs, the configuration parameters available in RCS 5.2 specification [1] will be extended with following additional provisioning parameters which control, by standardised means:

- the API access control policy of the MNO;
- the certificates that are trusted and used to support IARI Range Authorization processing.

IARI range configuration data is presented to the stack when service parameters are provisioned, as a single element contained within the OTHER/EXT subtree of the HTTP XML configuration structure.

8.2 Device Management parameters for API policy

Configuration parameter	Description	Notes
EXTENSIONS POLICY	This parameter indicates to a device the types of Extensions that are authorised to access the RCS infrastructure If this parameter is set to: 0, Only second party Extensions (i.e. MNO-trusted applications) are authorised to access and use the RCS infrastructure. An authorised second party Extension is an Extension which has a iari authorization signature that chains to one of the second party range certificates provided through the IARI Authorization Info parameter. An Extension without a valid iari authorization signature cannot access the stack. 1, Second party Extensions and third party Extensions are authorised to access the RCS infrastructure. For second party Extensions, similar procedures apply as for value 1. For third party Extensions, the app accessing the API shall have a IARI Authorization corresponding to its unique IARI. This parameter is not applicable to a device not compatible with the Extensions (e.g. not exposing terminal APIs) or if ALLOW RCS EXTENSIONS (defined in section A.1.16 of [1]) is set to 0	Optional Parameter Mandatory if ALLOW RCS EXTENSIONS (defined in section A.1.16 of [1]) is set to 1
IARI Authorisation Info	The IARI Authorization Info contains a list of IARI ranges and associated X509 certificates. Together, this information is to be used to validate on the device any signature in a IARI Authorization document for a IARI within the specified range. The range of IARIs is defined using a string ending with the '*' wildcard character. E.g. urn%3Aurn-7%3A3gpp-application.ims.iari.rcs .mnc001.mcc002.* An X509 Certificate element is an element whose text body is a Base64-encoded X509v3 certificate (base64 encoding as per RFC2045).	Optional Parameter Mandatory if EXTENSIONS POLICY is set.

Table 2: RCS API Extensions Policy configuration parameters

EXTENSIONS POLICY and IARI Authorization Info are placed in APIExt MO sub tree, located in the Ext node of the Other subtree defined in section A.2.10 of RCS 5.2 [1].

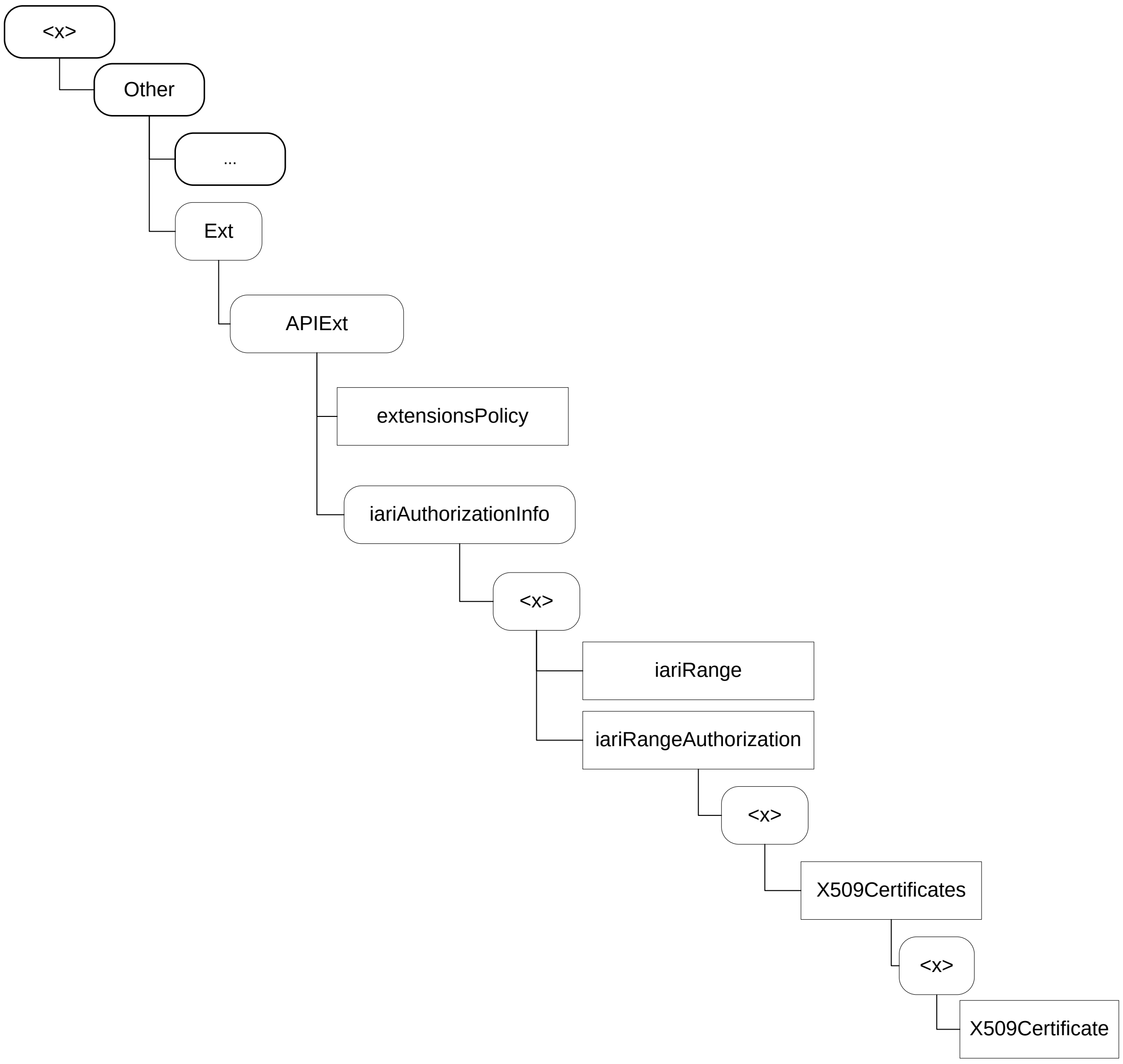


Figure 14: New APIExt sub tree in Other Ext MO

The associated HTTP configuration XML structure is presented in the table below:

```
<characteristic type=” APIExt”>
  <parm name=”extensionsPolicy” value=”X”/>
  <characteristic type=”iariAuthorizationInfo”>
       <characteristic type=” iariRangeAuthorizations”>
         <characteristic type=” iariRangeAuthorization1”>
              <parm name=”iariRange” value=”X”/>
              <characteristic type=”X509Certificates”>
                <parm name=”X509Certificate1” value=”X”/>
                <parm name=”X509Certificate2” value=”X”/>
                 ….
               </characteristic>
         </characteristic>
         <characteristic type=” iariRangeAuthorization2”>
              <parm name=”iariRange” value=”X”/>
              <characteristic type=”X509Certificates”>
                <parm name=”X509Certificate1” value=”X”/>
                <parm name=”X509Certificate2” value=”X”/>
                 ….
               </characteristic>
         </characteristic>
         ….
       </characteristic>
  </characteristic>
</characteristic>
```

Table 3: APIExt sub tree associated HTTP configuration XML structure

This structure will be included into the configuration document defined in section A.3 of RCS 5.2 [1] as follows:

```
<?xml version="1.0"?>
<wap-provisioningdoc version="1.1">
        <characteristic type="VERS">
              <parm name=”version” value=”1”/>
              <parm name=”validity” value=”1728000”/>
        </characteristic>
        <characteristic type="TOKEN">
              <parm name=”token” value=”X”/>
        </characteristic>
        <characteristic type="MSG">                    -- This section is OPTIONAL
              <parm name=”title” value=”Example”/>
              <parm name=”message” value=”Hello world”/>
              <parm name=”Accept_btn” value=”X”/>
              <parm name=”Reject_btn” value=”X”/>
        </characteristic>
        <characteristic type="APPLICATION">
              <parm name=”AppID” value=”ap2001”/>
              <parm name=”Name” value=”IMS Settings”/>
              <parm name=”AppRef” value=”IMS-Settings”/>
                    …
        </characteristic>
        <characteristic type="APPLICATION">
              <parm name=”AppID” value=”ap2002”/>
              <parm name=”Name” value=”RCS settings”/>
              <parm name=”AppRef” value=”RCSe-Settings”/>
              <characteristic type=”IMS”>
```

```
            <parm name=”To-AppRef” value=”IMS-Settings”/>
        </characteristic>
        <characteristic type=”SERVICES”>
            …
        </characteristic>
        <characteristic type=”PRESENCE”>
            …
        </characteristic>
        <characteristic type=”XDMS”>
            …
        </characteristic>
        <characteristic type=”SUPL”>
            …
        </characteristic>
        <characteristic type=”IM”>
            …
        </characteristic>
        <characteristic type=”CPM”>
            …
        </characteristic>
        <characteristic type=”CAPDISCOVERY”>
            …
        </characteristic>
        <characteristic type=”APN”>
            …
        </characteristic>
        <characteristic type=”OTHER”>

          <parm name=”endUserConfReqId” value=”X”/>

          <parm name=”allowVSSave” value=”X”/>

          <characteristic type=” transportProto”>

          <parm name=”psSignalling” value=”X”/>

          <parm name=”psMedia” value=”X”/>

          <parm name=”psRTMedia” value=”X”/>

          <parm name=”wifiSignalling” value=”X”/>

          <parm name=”wifiMedia” value=”X”/>

          <parm name=”wifiRTMedia” value=”X”/>

          </characteristic>

          <parm name=”uuid_Value” value=”X”/>

          <parm name=”IPCallBreakOut” value=”X”/>

          <parm name=”IPCallBreakOutCS” value=”X”/>

          <parm name=”rcsIPVideoCallUpgradeFromCS” value=”X”/>

          <parm name=”rcsIPVideoCallUpgradeOnCapError” value=”X”/>

          <parm name=”rcsIPVideoCallUpgradeAttemptEarly” value=”X”/>

          <parm name=”extensionsMaxMSRPSize” value=”X”/>

          <parm name=”maximumRRAMDuration” value=”X”/>
          <characteristic type=”Ext”>
              <characteristic type=” APIExt”>
                <parm name=”extensionsPolicy” value=”X”/>
                <characteristic type=” iariAuthorizationInfo”>
                     <characteristic type=” iariRangeAuthorizations”>
                       <characteristic type=” iariRangeAuthorization1”>
```

```
                                                   <parm name=”iariRange” value=”X”/>
                                                   <characteristic type=”X509Certificates”>
                                                     <parm name=”X509Certificate1” value=”X”/>
                                                     <parm name=”X509Certificate2” value=”X”/>
                                                       ….
                                                    </characteristic>
                                           </characteristic>
                                           ….
                                        </characteristic>
                                 </characteristic>
                              </characteristic>
                         </characteristic>

                   </characteristic>
                   <characteristic type=”SERVICEPROVIDEREXT”>
                             …
                   </characteristic>

          </characteristic>
</wap-provisioningdoc>
```

Table 4: Complete RCS HTTP configuration XML structure

The parameters for the API Policy are formally defined in the sections below.

Node: /<x>/Other/Ext/APIExt

Under this interior node the RCS parameters related to API policy are placed.

Status	Occurrence	Format	Min. Access Types
Optional	ZeroOrOne	Node	Get

Table 5: Other MO sub tree APIExt node

- Values: N/A
- Type property of the node is: urn:gsma:mo:rcs-other:5.2:Ext:APIExt
- Associated HTTP XML characteristic type: “APIEXT”

Node: /<x>/Other/Ext/APIExt/extensionsPolicy

Leaf node that describes the types of Extensions authorized by the MNO to access the RCS infrastructure.
It is required to be instantiated if allowRCSExtensions is set to 1.

Status	Occurrence	Format	Min. Access Types
Required	ZeroOrOne	Int	Get

Table 6: APIExt MO sub tree addition parameters (extensionsPolicy)

- Values:

  0 - Only second party Extensions (MNO trusted applications) are authorized to access and use the RCS infrastructure. A second party Extension is an Extension which has an IARI Authorization signature that chains to one of the second-party

certificates sent through the IARI Authorization Info. An Extension without a valid IARI Authorization cannot access the stack.
1 – Second-party Extensions and third-party Extensions are authorized to access the RCS infrastructure. For second-party Extensions, similar procedures apply as for value 1. For third-party Extensions, the app accessing the API shall have an IARI Authorization corresponding to its unique IARI.

- Post-reconfiguration actions: The client should be reset and should perform the complete first-time registration procedure following a reconfiguration (e.g. OMA-DM/HTTP) as described in section of RCS 5.2 [1].
- Associated HTTP XML parameter ID: “extensionsPolicy”

Node: <x>/APIExt/iariAuthorizationInfo

A Placeholder interior node for the configuration related to the information requested to authorize MNOs Extensions.
It is required to be instantiated if extensionsPolicy is set.

Status	Occurrence	Format	Min. Access Types
Required	ZeroOrOne	Node	Get

Table 7: APIExt MO sub tree iariAuthorizatioInfo node

- Values: N/A
- Type property of the node is: *urn:gsma:mo:rcs-other:5.2:Ext:APIExt:iariAuthorizationInfo*
  Associated HTTP XML characteristic type: “iariAuthorizationInfo”

Node: /<x>/Other/Ext/APIExt/iariAuthorizationInfo/<x>

A Placeholder interior node where to place 1 or more iariRangeAuthorization nodes.

Status	Occurrence	Format	Min. Access Types
Required	One	Node	Get

Table 8: APIExt sub tree iariRangeAuthorizations node

- Values: N/A

  Type property of the node is: urn:gsma:mo:rcs-other:5.2:Ext:APIExt:iariAuthorizationInfo:iariRangeAuthorizations

- Associated HTTP XML characteristic type: “iariRangeAuthorizations”

Node: /<x>/Other/Ext/APIExt/iariAuthorizationInfo/<x>/iariRangeAuthorization

An interior node where to place 1 or more iariRangeAuthorization nodes.

Status	Occurrence	Format	Min. Access Types
Required	ZeroOrMore	Node	Get

Table 9: APIExt sub tree iariRangeAuthorization node

- Values: N/A

  Type property of the node is: urn:gsma:mo:rcs-other:5.2:Ext:APIExt:iariAuthorizationInfo:iariRangeAuthorizations:iariRangeAuthorization

- Associated HTTP XML characteristic type: “iariRangeAuthorization<X>” where <X> is a positive integer value

Node: /<x>/Other/Ext/APIExt/iariAuthorizationInfo/<x>/iariRangeAuthorization/iariRange

The <IARIRange> node is a leaf node contains a range of IARIs authorised by the MNO through the certificates contained in the associated <X509Certificate> leaf nodes.

Status	Occurrence	Format	Min. Access Types
Required	One	String	Get

Table 10: APIExt sub tree addition parameters (iariRange)

- Values: range of IARIs authorized by the MNO through the certificates contained in the associated <X509Certificate> nodes. The range of IARIs may contain only one IARI or several. In the latter case it is defined using a string ending with the '*' wildcard character. E.g. urn%3Aurn-7%3A3gpp-application.ims.iari.rcs.mnc001.mcc002.*
- Post-reconfiguration actions: The client should be reset and should perform the complete first-time registration procedure following a reconfiguration (e.g. OMA-DM/HTTP) as described in section 2.3 **Error! Reference source not found.**of RC 5.2 [1].
- Associated HTTP XML parameter ID: “iariRange”

Node: /<x>/Other/Ext/APIExt/iariAuthorizationInfo/<x>/iariRangeAuthorization/<x>

A Placeholder interior node where to place 1 or more X509Certificate leaves.

Status	Occurrence	Format	Min. Access Types
Required	One	Node	Get

Table 11: APIExt sub tree X509Certificates node

- Values: N/A

  Type property of the node is: urn:gsma:mo:rcs-other:5.2:Ext:APIExt:iariAuthorizationInfo:iariRangeAuthorization:X509Certificates

- Associated HTTP XML characteristic type: “X509Certificates”

Node: /<x>/Other/Ext/APIExt/iariAuthorizationInfo/<x>/iariRangeAuthorization/<x>/X509Certificates/<x>/X509Certificate

Status	Occurrence	Format	Min. Access Types
Required	One	String	Get

Table 12: APIExt MO sub tree addition parameters (X509Certificate)

- Values: Base64-encoded X509v3 certificate (base64 encoding as per RFC2045). The encoded certificate must be checked for syntactic validity; any invalid certificate causes the entire enclosing iariRange to be invalid.
- Post-reconfiguration actions: The client should be reset and should perform the complete first-time registration procedure following a reconfiguration (e.g. OMA-DM/HTTP) as described in section 2.3 of RC 5.2 [1].
- Associated HTTP XML parameter ID: “X509Certificate”

Associated HTTP XML characteristic type: “X509Certificate<X>” where <X> is a positive integer value

Annex B Document Management

B.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	02 May 2014	Initial version for approval, new PRD RCC.55	RCSJTA	P. Byers, D. O’Byrne, Kelvin Qin / GSMA

B.2 Other Information

Type	Description
Document Owner	RCS RCSJTA
Editor / Company	Paddy Byers, David O’Byrne, Kelvin Qin / GSMA

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com.

Your comments or suggestions & questions are always welcome.