RCS Extensibility, Terminal and Network API Security

Version 2.0

23 June 2015

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

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is confidential to the Association and is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Antitrust Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

Table of Contents

1 Introduction

1.1 Overview

This note outlines a proposal to support extensible RCS, looking at security issues for the Terminal and Network APIs and the device stack, and how those extensions are managed.

NOTE: The approach set out in this document is in response to an IP Communications Leadership Team 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: This document relates to the RCS 5.3 Specification and not any earlier versions of the RCS specification.

NOTE: This document identifies a proposed architecture for the security mechanisms and processes working for Terminal and Network APIs. The architecture identifies a requirement for a developer / application repository as a common capability which shares out information to operator systems to support the various provisioning and run-time processes[1].

1.2 Definitions

Term	Description
ICSI	IMS Communication Service Identifier
RCS	Rich Communications Services
RCS extensions	Applications adding functionality to native devices utilising RCS APIs
ISV	Independent Software Vendor
IARI	IMS Application Reference Identifier (IARI)
T-API	Terminal RCS API
N-API	Network RCS API
MSRP	Message Session Relay Protocol

1.3 References

Ref	Doc Number	Title
[1]	[RCS 5.3]	GSMA PRD RCC.07 - RCS 5.3 - Advanced Communications Services and Client Specification, Version 6.0, 28 Feb 2015 http://www.gsma.com/network2020/wp-content/uploads/2015/03/RCS5.3_UNI.zip
[2]	RFC 2119	“Key words for use in RFCs to Indicate Requirement Levels”, S.

[1] The suggested model for such a centralized repository is the GSMA API Exchange enhanced for RCS applications through the addition of information relating to Feature Tags/ IARIs.

Ref	Doc Number	Title
		Bradner, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

1.4 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 API Security

2.1 Context

As part of the Network 2020 Programme, GSMA is supporting member operators in launching commercial deployments of Rich Communication Services (RCS), enabling applications and services running on devices with native RCS stacks to be delivered over the IP Multimedia Subsystem (IMS) infrastructure. The RCS 5.3 specification defines a range of services including enriched call, messaging, multimedia and other services, which are natively available on RCS enabled handsets.

However, operators require additional services to be created and deployed 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 RCS APIs, both Terminal APIs (T-API) available to applications on the handset and Network APIs (N-API) in the network and deploying applications, created by Terminal Device OEMs, MNOs and third party ISVs. As part of this approach, it is expected that there should be interworking between Terminal APIs and Network APIs including the use of the service extension framework and security model.

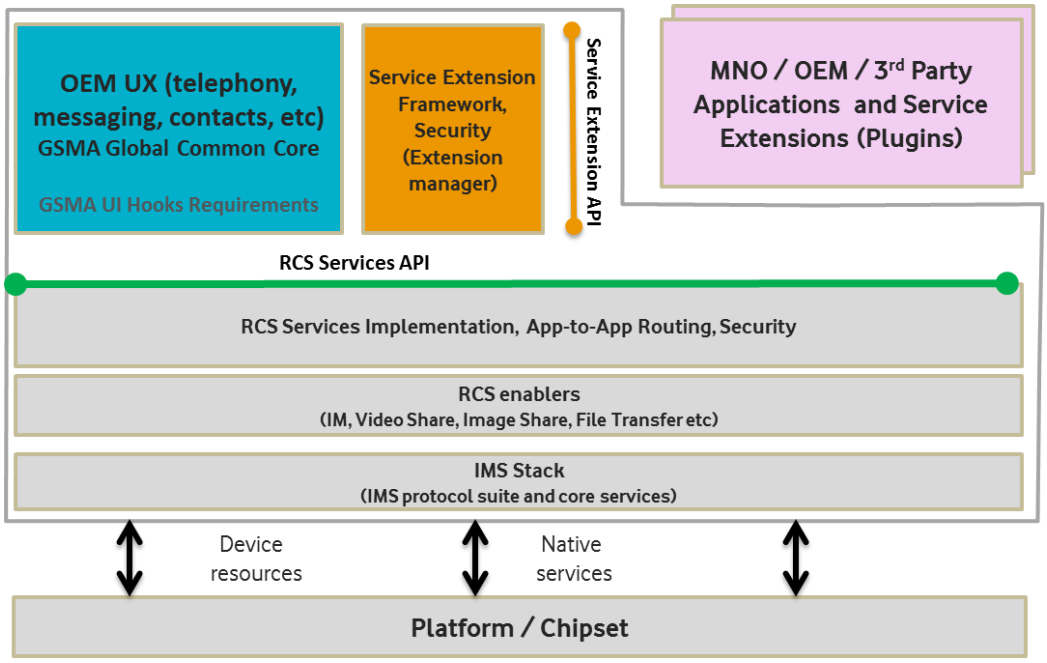

Figure 1: General Terminal API Architecture Overview

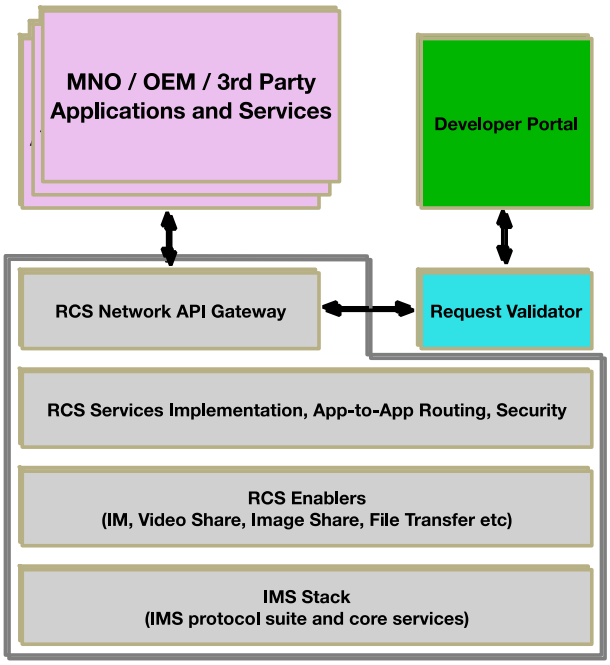

Figure 2: General Network API Architecture Overview

RCS services are considered to be extensible in that new applications and services can be created on top of the fundamental RCS protocols (chat, file transfer etc.). 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 traffic to be

handled appropriately in the network and also, routed correctly to the correct T-API or N-API client application. Tags are also the basis on which one device can discover which applications or services are available on peer devices. This extensibility feature means that later revisions of the RCS standards can incorporate new services as they are developed.

The current generation of social messaging apps, shows the potential of this approach; for example, 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 minimally-intrusive way,
- Ensuring the security framework requires no single controlling/administrative organisation.

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 security,
- Section 3 details the application identification, and its management by the operators so that applications gain access to RCS APIs,
- Section 4 discusses three principal API access use cases and states their related high-level requirements,
- Section 5 outlines the process for managing Terminal API access for applications, and for controlling their corresponding tags,
- Section 6 outlines the process for managing Network APIs, and for controlling their corresponding tags and API credentials,
- Sections 7 and 8 identify the concrete steps needed to implement the proposal, addressing additions to RCS specifications and terminal and infrastructure implementations,
- Section 9 defines the device provisioning process dedicated to API.

2.3 Scope

2.3.1 What is enabled by the proposed approach

T-API access:

- Access by any app to all RCS T-APIs,
- Enablement of apps using the T-APIs for core RCS services (i.e. chat, file, image and video share) and which interoperate with the native UI for core services on other RCS-enabled devices,

- Revocation by the service provider (MNO) of access by specific applications (designated via IARI's) to RCS T-APIs.

N-API access:

- Creation of network apps which use “Open” APIs (chat, file transfer, etc.) delivered via HTTP RESTful API definitions relating to SIP addresses. These are considered to be non-sensitive, or of low sensitivity such that the user can authorise access. (see section 4.3.1),
- Control by the service provider (MNO or MVNO) over which network applications can interact with terminal applications using controlled T-APIs including core service APIs on the devices subscribed to their network,
- Revocation by the service provider (MNO) of access by specific applications (designated via IARI's) to RCS N-APIs.

NOTE: There may be different methods employed by MNOs for the way in which applications using Network APIs are verified against the associated IARI Authorisation document. This document does not prescribe any specific approach i.e. it could be a part of the IMS infrastructure, N-API gateway or provided by a third party organisation. The assumption is, this process can in future scale to support full 3rd party access to T-APIs and N-APIs. The workflow and processes supporting that are out of scope.

RCS extensions:

- Enablement of MNO and Third party T-API and N-API applications for which unique custom feature tags are created by the developer,
- Identification of T-API applications on a device implementing RCS extensions during Capability Discovery,
- Identification of the traffic pertaining to a specific RCS extension, for app-to-app traffic routing,
- Prevention of apps from using the unique custom feature tag belonging to any other extension without getting prior approval from the creator of that tag,
- Revocation by an individual service provider (MNO) of access by specific applications (designated by IARI) to RCS extension APIs for devices or network applications subscribed to that provider’s network,
- Global revocation of access by specific applications (designated by IARI) to sensitive RCS extension APIs across all networks.

2.3.2 What is not enabled by this approach

- Preventing an app on a jail-broken or hacked device from using another app’s unique custom tag,

- Preventing a rogue network API app from masquerading as a valid network API app in the case that the credentials of the valid network API app become known to an attacker[2],
- Revocation of apps in the sense of removing them from end users devices is not supported
- Differentiated access to APIs; an application, if authorized to use core service APIs, can use all core service APIs,
- Ability to retrospectively add a new core service to the stack. (NOTE: A proposed approach for may submitted as a CR for a possible maintenance release of RCS 5.3).

2.4 Supported Use Cases

NOTE: That in all cases below 'application' (or 'app') is considered to be uniquely identified by permitted access to a specific IARI.

2.4.1 Terminal-API Use Cases

Case #	Title	Description
T-1	Service Integration by Device OEM at manufacture	The OEM will integrate the RCS stack and conforming T-API implementation, as well as the app accessing the T-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 T-API.
T-2	Service Addition by end user	User can install an app that offers an RCS-based communication service, using on-device RCS stack, coexisting with existing apps. App can send messages to the same app on other devices. MNO can block traffic on its own network from an app without affecting core services from native devices or other apps.
T-3	Addition of app for core services by end user	User can install an app that offers an RCS-based communication service, using on device RCS stack, co-existing with existing apps. App can send messages to the native app for that traffic type on other devices. MNO can block traffic on its own network from such an app without affecting core services from native devices or other apps.
T-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”).
T-5	Service discoverability [App	Installed T-API apps are discoverable to the end users

[2] Additional security mechanisms should be employed by the MNO for example restricting network API requests to approved origins via IP address whitelisting, request throttling and the practice of initiating network API requests from secured servers rather than insecure clients e.g. web clients.

Case #	Title	Description
	discoverability]	contacts using the RCS capability discoverability method (such discoverability can be filtered by MNO).
T-6	App to app traffic routing	A message sent from an RCS-enabled app or service extension can be received directly in the B-party's app (i.e. app to app) instead of being routed to the RCS inbox. Every app is identified by an IARI which is globally unique. Traffic can be blocked on network by IARI. Apps can be blocked from accessing T-API 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 and user-to-developer communications.
T-7	T-API access control & revocation	MNOs can verify at the network whether a specific IARI is associated with an app that has been flagged as malicious or blocked for some other reason. MNOs can revoke access to the APIs for a specific application for all devices. MNOs can block traffic by IARI (at IMS/application gateway).
T-8	Download communications app, RCS as a bearer	Third parties can develop their communications apps to use RCS as a transport mechanism.
T-10	Controlled discoverability	When the application is using application to application traffic, it will require discoverability process. For these applications, the end user: • Can see from within an app and the contact card which of his/her contacts have the same app, • Is prompted at app install to select whether the app should be discoverable or not, • Can opt to have an app not be discoverable, • Can only see the apps on the B party device that they have in common and are permitted by the B party to be discoverable, • Apps are no longer discoverable once deleted.
T-11	Usage tracking	The MNO shall be able to discover how many instances of each app are active on their network.
T-12	Traffic measurement	The MNO shall be able to deploy a network element to measure the IARI-specific traffic.

Table 1: Supported T-API Use Cases

2.4.2 Network-API Use Cases

Case #	Title	Description
N-1	Service Addition or Invocation by end user	An app may offer an RCS-based communication service, accessing the RCS network via the N-API. App can send messages to the other instances of the

Case #	Title	Description
		same app, accessing the network either by N-API or T-API. MNO can block traffic on its own network from an app without affecting core services from native devices or other apps.
N-2	Service discoverability [App discoverability]	Known N-API apps should be discoverable based on IARI and the context of the OMA Network APIs. Work may be required with OMA to develop standard to support discoverability. Additional standard works are out of the scope of this document.
N-3	App to app traffic routing	A message sent from an RCS-enabled app or service extension can be received either in the RCS inbox OR directly in the B-party's app. Every app is identified by an IARI which is globally unique. Traffic can be blocked on network by IARI. Apps can be blocked from accessing N-API by IARI. ISVs can add discoverability and in-app communications to their own non-communications apps for viral distribution, recommendation-based distribution, social gaming and user-to-developer communications.
N-4	N-API access control & revocation	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. The MNO can verify the app is presenting the correct authorisation/ identification credentials (client ID) associated with the IARI via an IARI authorisation document. MNOs can revoke access to the N-API for a specific application (designated by IARI) for all devices. MNOs can block traffic by IARI (at IMS/application gateway).
N-5	Download communications app, RCS as a bearer or use for example a web app which uses RCS as a bearer (e.g. customer care)	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 N-APIs. Applications are identified by IARI using processes defined for creating IARI's as per for T-API applications.
N-6	Adherence to Terms and conditions	The MNO will normally require the 3rd party developer to sign up to terms and conditions as a pre-requirement before being granted access to N-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 permitting traffic from the app.
N-7	Controlled discoverability	The end user can see from within an app and the contact card which of his/her contacts have the same app subject

Case #	Title	Description
		to the following: • Is prompted at app install or first run to select whether the app should be discoverable or not, • Can opt to have an RCS app not be discoverable, • Cannot see all the apps on the B parties device, just the ones they have in common and which are discoverable, • Sees apps which may only be installed on the B party's secondary devices, • Apps are no longer discoverable once deleted.
N-8	Usage tracking	The MNO shall be able to discover how many instances of each app are active on their network.
N-9	Traffic measurement	The MNO shall be able to deploy a network element to measure the IARI-specific traffic. The MNO shall be able to deploy a network element to measure the N-API specific traffic for a given application (identified using client ID). The MNO shall be able to (if required) charge the developer/publisher for traffic usage by an application based on N-API specific traffic.

Table 2: Supported N-API Use Cases

3 Application Identification and Operator Control

3.1 Introduction

This section summarises the mechanisms for application identification and access control to RCS services, which underpin the other mechanisms described in the later sections.

Application identifiers (IARIs) are used for four specific purposes.

The first is to make apps discoverable. The IARIs of T-API applications on a device, or N-API applications known to the network are returned as part of Capability discovery, allowing peer apps to know that a contact has the same (or compatible) app.

The second purpose is to identify traffic from a specific application to the RCS stack and to the network. For T-API, the application identifier is in the Contact header of all SIP 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. For network based applications, the application identifier for requests passed over HTTP is based on client ID along with a permitted IARI relevant to the request.

The third use of application identifiers is to enable peer apps to create private "virtual networks" in which RCS are used privately between two identical applications or peers, on different devices. Those peers, if they are each authorised to use a specific private IARI, 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 use a transparent peer-to-peer MSRP service or 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.

The final purpose of application identifiers is to allow, when circumstances demand, the traffic from that application to be blocked. This can be performed in the network and, in the case of T-API (RCS 5.3), the stack on the device can be notified of a blocked IARI which prevents the associated app from accessing the API. A block in the network may be enforced at the UNI or at the NNI (which applies in the case that a particular IARI is permitted in one network but blocked in an adjacent network).

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

3.2 IARI structure

The IARI structure to be used for RCS service extensions is detailed in section 2.6.1.1.3 of [1].

3.3 Self-signed IARIs

The access control mechanisms described in this document are based on a space of IARIs designated for use by “extensions”, i.e. applications or services that are not assigned IARIs within existing specifications. Such extension IARIs are supportable without the need for any central issuer of identities (i.e. of IARIs, or apps, or developers). This enables third-party and MNO developers to be free to 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 or reliance on a central issuing authority.

Self-signed IARIs are defined in that specification as a specific range in the IARI range for Third Party Extension (defined in section 2.6.1.1.3 of [1]) 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. Once a given application is authorized by a tag owner to use that tag, that authorization is captured as a digital signature; a Tag Authorisation document 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.

For T-API apps, this authorisation can be packaged as part of the application. Verification of that signature, confirming for T-API to the stack that the application is authorized to use that tag, is performed by validating the signature in the Tag Authorisation. The process related to an independent T-API app incorporating the IARI authorisation is detailed in section 5.

The equivalent process for a Network API based application is detailed in section 6.

3.4 Application identification at the SIP protocol level

Each RCS session operating over SIP is 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 (see section 3.12.4 of [1). 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].

Network APIs use HTTPS (over TCP/IP) as a bearer for the requests, API calls use OMA defined RESTful interfaces. The Network API gateway ensures the conversion of application identifiers between HTTP headers to/from SIP Contact and Accept-Contact headers as required. This process, therefore, requires the Network API gateway to have access to application identifiers for Network API requests and IARI's for T-API requests. This process is detailed in Section 6.

3.5 MNO models for application identification management

In the following architecture and related description, these terms are used:

- Federation – a loosely coupled group of operators who have established the framework for applications across the federation to be used by mobile users on any of the networks participating in the federation,

- Serving Operator – this is the MNO who is processing an RCS API call for one of their end users,

- Developer Operator – this is an MNO who operates a developer programme and introduces the developer/ application to the federation. Usually each Developer Operator is also functioning as a Serving Operator, though not every Serving Operator has to function as a Developer Operator,

- Central Developer/Application Repository – this is a logical component which maintains key information about developers and applications and manages the replication of this information to federated operators (Local Cache),

- Local Cache Repository – provided by Serving Operators, or on their behalf, to hold information on developers/applications (including IARI, tag authorisation document and client ID) and globally relevant status,

- API Validator – provided by Serving Operators, or on their behalf, is the functional unit that decides if an RCS API call is valid for processing based on all appropriate factors considered by the operator. This at least considers the data from the developer/ application registry including IARI and tag authorisation document. In addition the API Validator can consider factors such as application approval state, Operator Terms & Conditions acceptance etc.,

- Authorisation Gateway – provided by Serving Operators, or on their behalf, is the functional unit that manages the authorisation procedures for RCS traffic:

    - IARI authorisation function: validate that the traffic identified by a specific IARI is authorised on the RCS network. The Authorisation Gateway will extract the IARI from either the SIP or HTTP signalling, depending on the location of the equipment that implements this function. It will normally refer to the 'API Validator' which has the responsibility for checking IARI/ IARI Authorisation and other factors,

    - Client ID authorisation function: in case of N-API application, validate that a specific application identified by its Client ID is authorised to generate RCS traffic for a specific IARI. The Authorisation Gateway will extract the Client ID and IARI from either the HTTP signalling. This function shall be implemented by the Operator's RCS N-API Gateway equipment.

- Client ID – used to identify Network API applications. Refer to section 6 for more information.

Two models are considered for application identifier (Client ID/IARI) distribution, depending on MNOs business strategy:

1- Independent MNO model:

    - Each MNO is responsible for the management of application identifiers to provide access to its own RCS network. There is no interaction between MNOs about identifier synchronization.

    - In N-API cases, the developer would register their application with each MNO they wish to work with, and be provided in return with a separate Client ID for each.

    - This approach is most typically adopted only for applications created by a MNO for its own customer base or a small number of local market enterprise customers

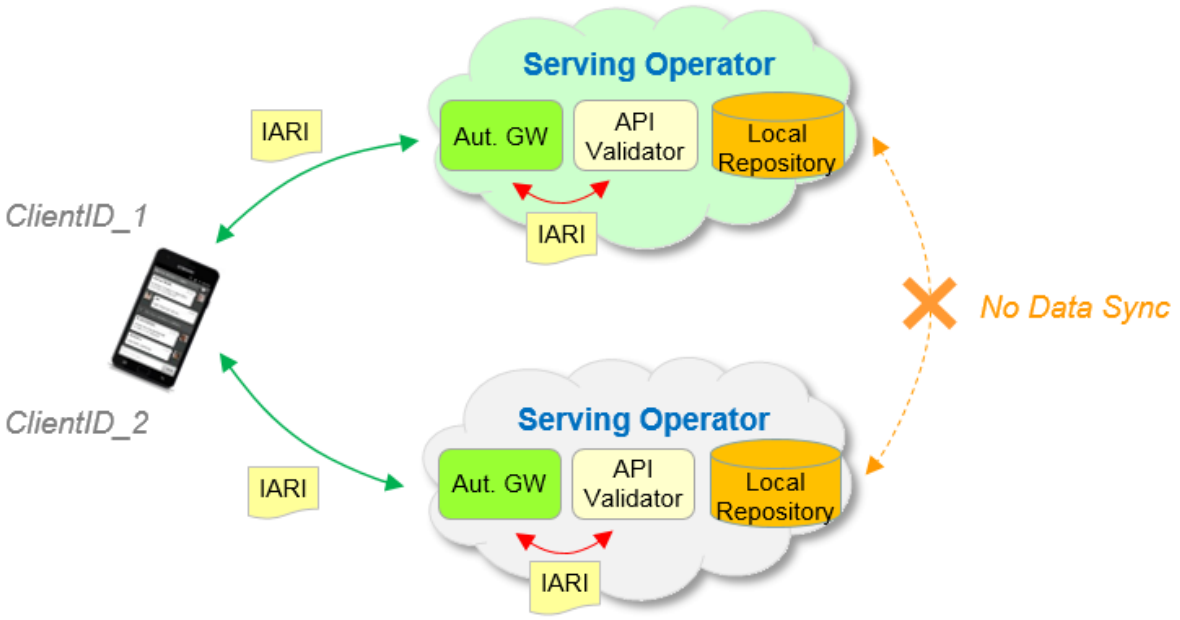

Figure 3: Independent MNO Model

2- Federated MNO model:

- IARI white/black lists are distributed between MNOs for interoperability.
- In N-API cases, the developer registers the application once and details are distributed between MNOs. Client IDs are either allocated according to a scheme to guarantee global uniqueness or otherwise synchronized between MNOs for interoperability, so that the same Client IDs can be used on various RCS networks. In such a case, the developer would register their application with a single MNO and be provided with a unique Client ID that could be used on various RCS networks, provided that the MNOs synchronize their Client ID namespaces or identifier allocations between each other.
- The identifiers (IARI / Client ID) storage is centralized in a shared repository. Its content is replicated in each MNO local repository, so that there is no hard dependency with the centralized database. This approach is most likely to be adopted for third-party applications for regional customer base served by multiple MNOs.

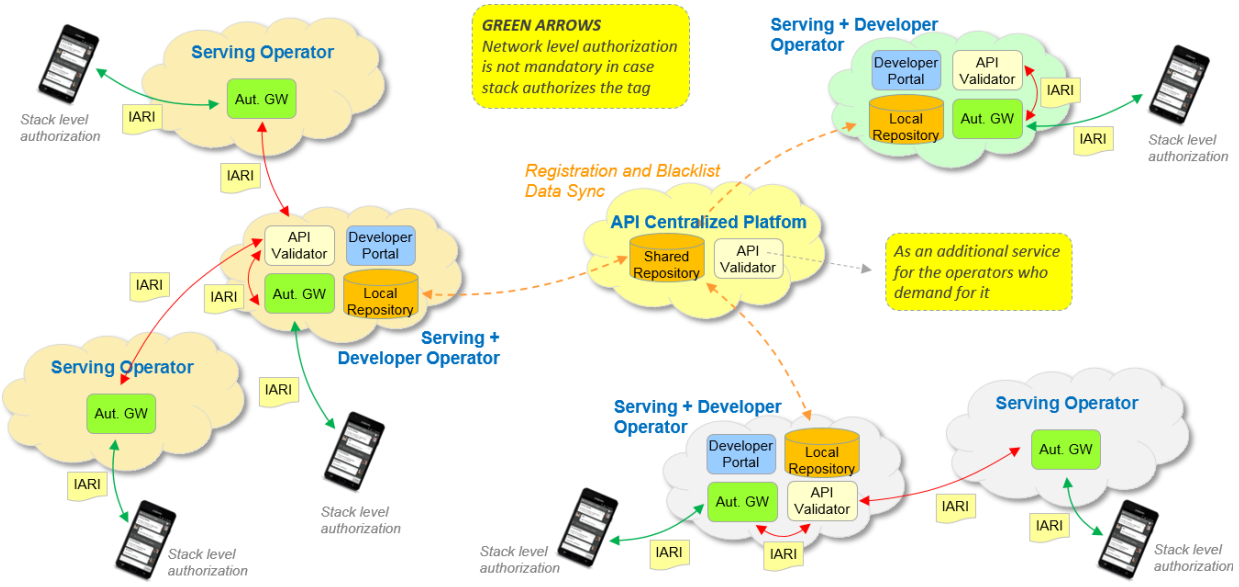

Figure 4: Federated MNO Model

NOTE This architecture supports both Terminal and Network API based applications, the respective infrastructure (IMS infrastructure or Network API gateway) uses the API Validator to check that API calls are permitted.

3.6 Developer / Application Registration

The mechanisms described in this document 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 business requirements relating to developer identification and there may be a requirement in certain circumstances for developers to enter into terms of service or commercial license agreements. This means that there would be some process that establishes a link between registered developer details and independently created 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.

It is expected that developer/application registration is decided individually by MNOs regarding whether this is required and if it is required exactly how this will work, e.g. does the MNO provide a developer portal. A typical process is outlined below:

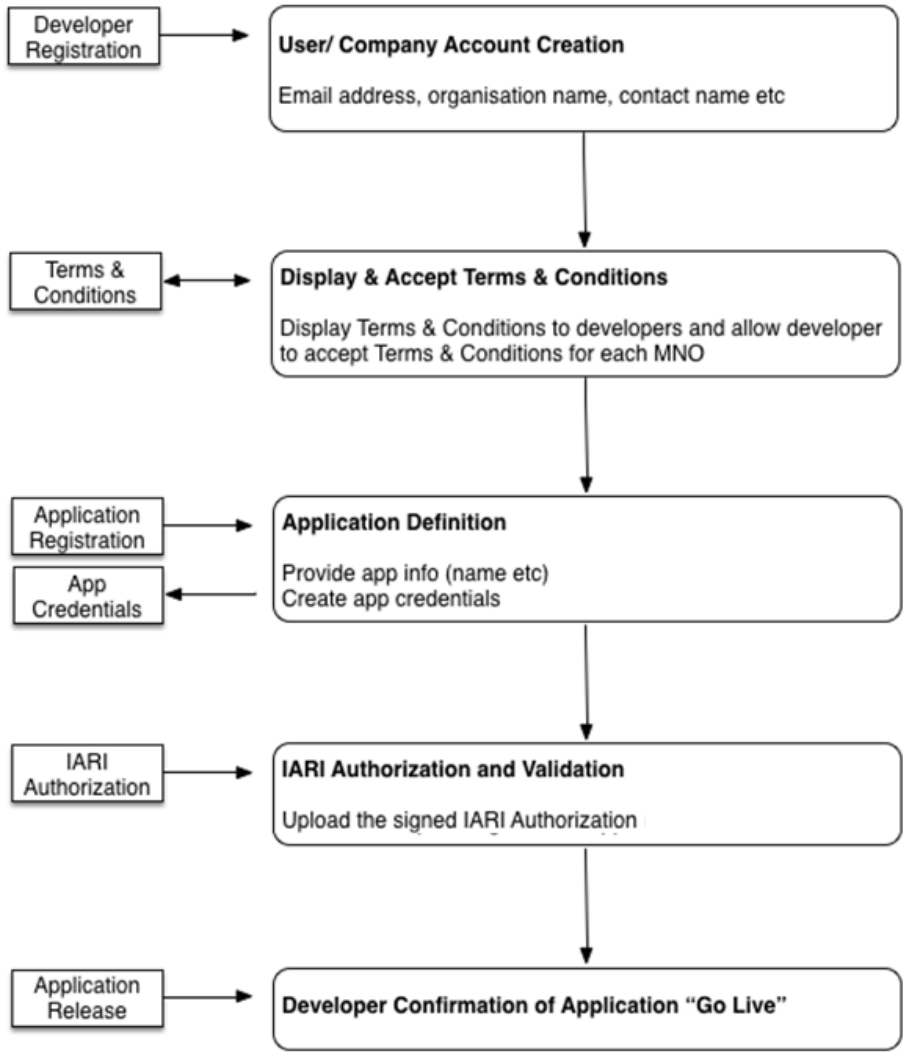

Figure 5: Developer/ Application Registration and IARI Upload

It is expected that the processes implemented for developer/application registration include:

- Accepting Terms & Conditions for the MNO(s) the developer wishes to partner;
- Providing any required information about the application that will help the MNO decide if the application should be approved according to their policies;
- Uploading the IARI Authorisation for the application which confirms the association between the application and the IARI used;
- In case the application requires use of Network APIs, there would be a process for issuing client ID and secret to developers which are used for HTTP request authorisation. The client ID would be included in the IARI Authorisation for the application which confirms that a requesting client ID is correctly associated with IARI;
- The developer confirming they want to release their application for production use.

4 Covered API Access Use Cases

4.1 Introduction

This section sets out the proposed approach to access control for the terminal and network APIs.

4.2 Scenarios

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

Access to T-APIs for core services by "native" apps integrated at manufacture. The OEM will integrate the RCS stack and conforming T-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 T-API.

Access to APIs for core services by non-native apps. This use-case applies to those apps acting as an RCS client giving access to one or more core services, possibly as part of a broader service offering. These might be T-API apps (i.e. as a replacement for the native RCS client) or N-API apps (e.g. for desktop or Wi-Fi only tablet RCS access).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 RCS extensions by apps. This use-case applies to apps using RCS to establish private app-to-app communication using a custom IARI.

4.3 API sensitivity

RCS APIs expose the ability to initiate and consume sessions for various RCS core services. As with many platform APIs, these may be open to abuse, for example, exposing the user to risks of cost or privacy, if they are made freely available to third-party apps. For this reason, the T-APIs require install-time approval in the same way as for all Android permissions at the dangerous protection level. In the case of N-API apps there may not be an equivalent 'install-time' procedure, so this means that the application must have an identity and associated credentials that are capable of revocation/blocking by a service provider (MNO).

However, the potential risks of abusive apps extend beyond the user. 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, access by any app to the service may therefore be additionally mediated by the service provider (MNO), based on a validated IARI; this is a real time access check, made by the network both for T-API and N-API access. As the check is online, and the check is made against a request validation service provided by the MNO, the MNO has ultimate control over whether or not any given API access request is permitted.

4.4 Extensions policy

The infrastructure access control mechanisms described above may not be universally deployed by all MNOs; this means that an MNO must be entitled to place a blanket restriction on running any non-native RCS applications on an attached device.

A mechanism is therefore defined whereby a service provider can indicate, as part of the configuration data for the service provisioned to each terminal, whether or not non-native applications are permitted to access the T-API. The details of this *extensions policy* setting are set out in section 9.

5 Terminal API

5.1 Introduction

This section sets out the proposed approach to access control for the terminal APIs, including the mechanisms proposed for independent authorisation of apps.

5.2 Self-signed application identification on T-API

This case covers the authorisation of applications using self-signed IARIs. Such applications are supportable 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.

Self-signed tags have the format:

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

The application-specific string is independently generated and unique. A Tag Authorisation document 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 authorized the app.

The process (showing a T-API app incorporating the authorisation for the self-signed IARI) is illustrated in the following figure.

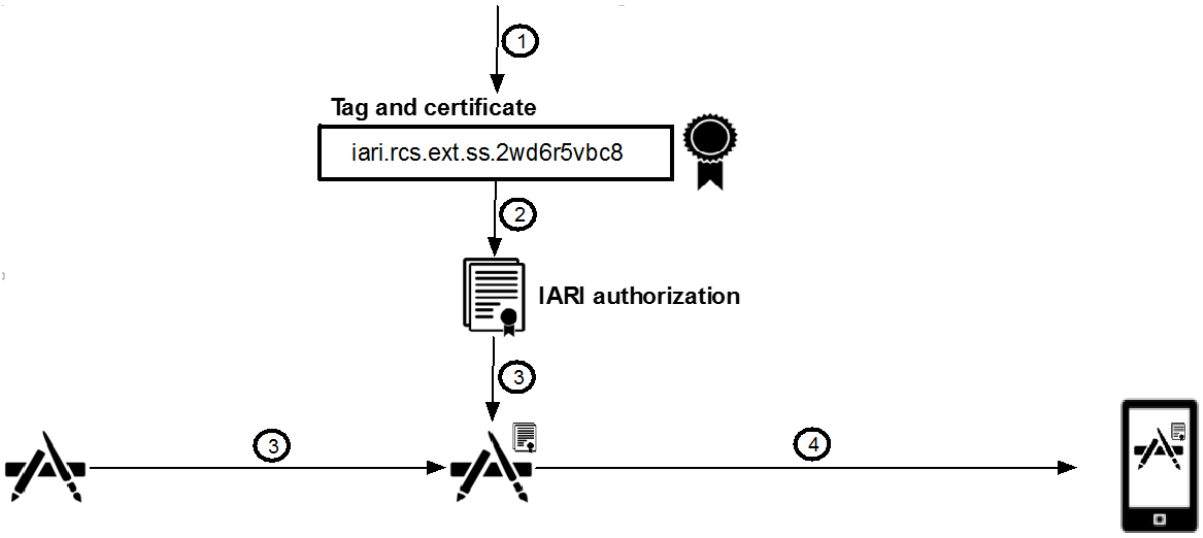

Figure 6: Applications with self-signed IARI (T-API)

- The tag owner generates the unique tag string and also a tag certificate.

- The tag owner creates a signed Tag Authorisation associating that tag with that app.
- The Tag Authorisation is packaged as part of the app by the application developer or publisher.
- Upon installation, the RCS stack verifies that the app is authorised to use that tag.

5.3 Self-signed tag management

5.3.1 Overview

Self-signed IARIs belong to a single “ext.ss” tag range for tags, and the tag generation process ensures that unique tag strings are generated.

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 key pair. 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 (who may be the same entity as the tag owner) must be granted authorization by the tag owner; this is captured in a signed IARI Authorisation document that references the application identity (developer’s release certificate and package name) and is signed by the tag owner using their private key. The developer writes his app in the normal way, presenting the IARI Authorisation document (in the application package), and uses standard procedures to publish the app.

This means that it is possible for the RCS environment 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 application identity and credentials. 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 tag owner, in just the same way as happens already for any other developer credentials.

Neither the tag owner or developer is identified by the signature; the chain of tag, IARI Authorisation and package signature only assures that the developer was authorised by the owner of the tag string to use that 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 T-API stack (applying a blacklist of tags) is specified.

The individual steps explained in greater detail in the following sections.

5.3.2 Tag owner creates tag

A public/private key pair 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 a 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 key pair 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.

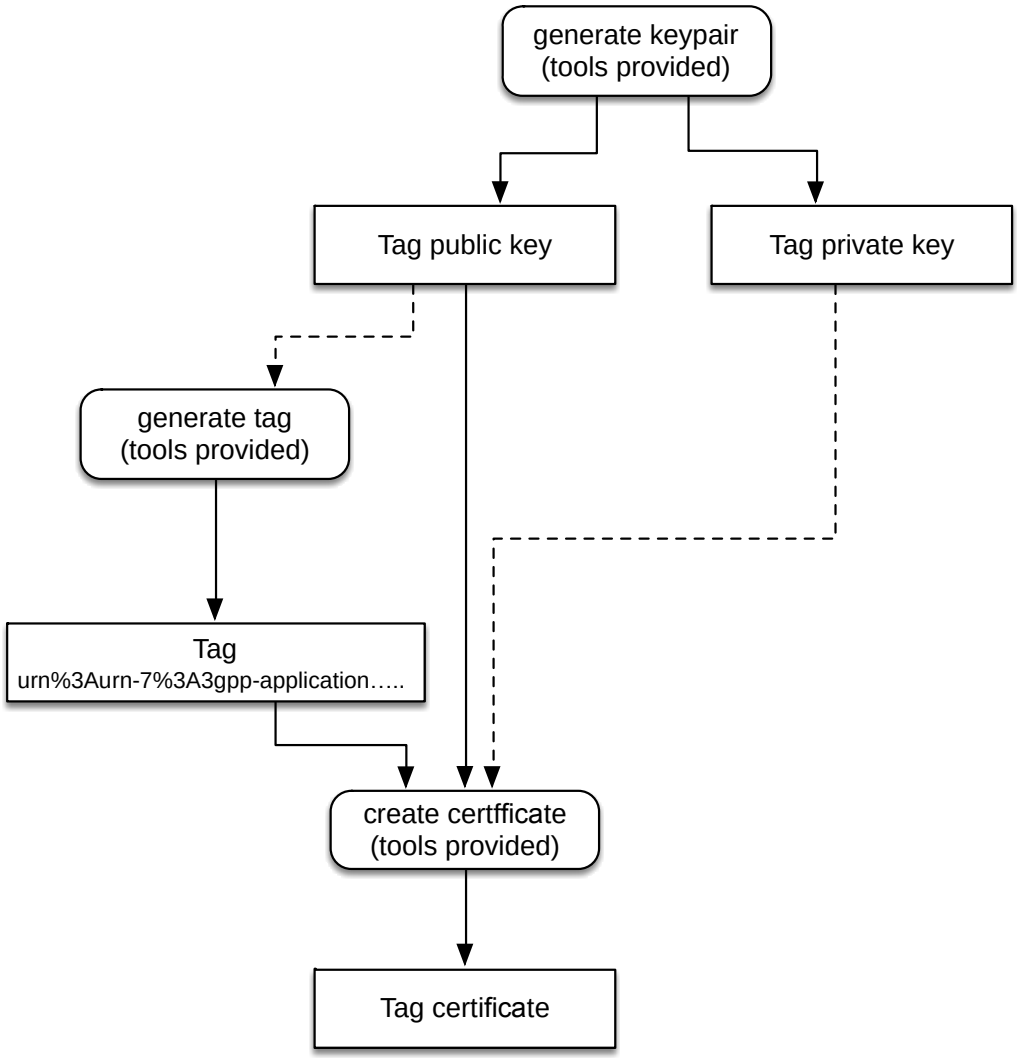

Figure 7: Tag owner creates tag

5.3.3 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 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 is routine for any application developer of Android applications for public distribution.

The process is illustrated in the figure below.

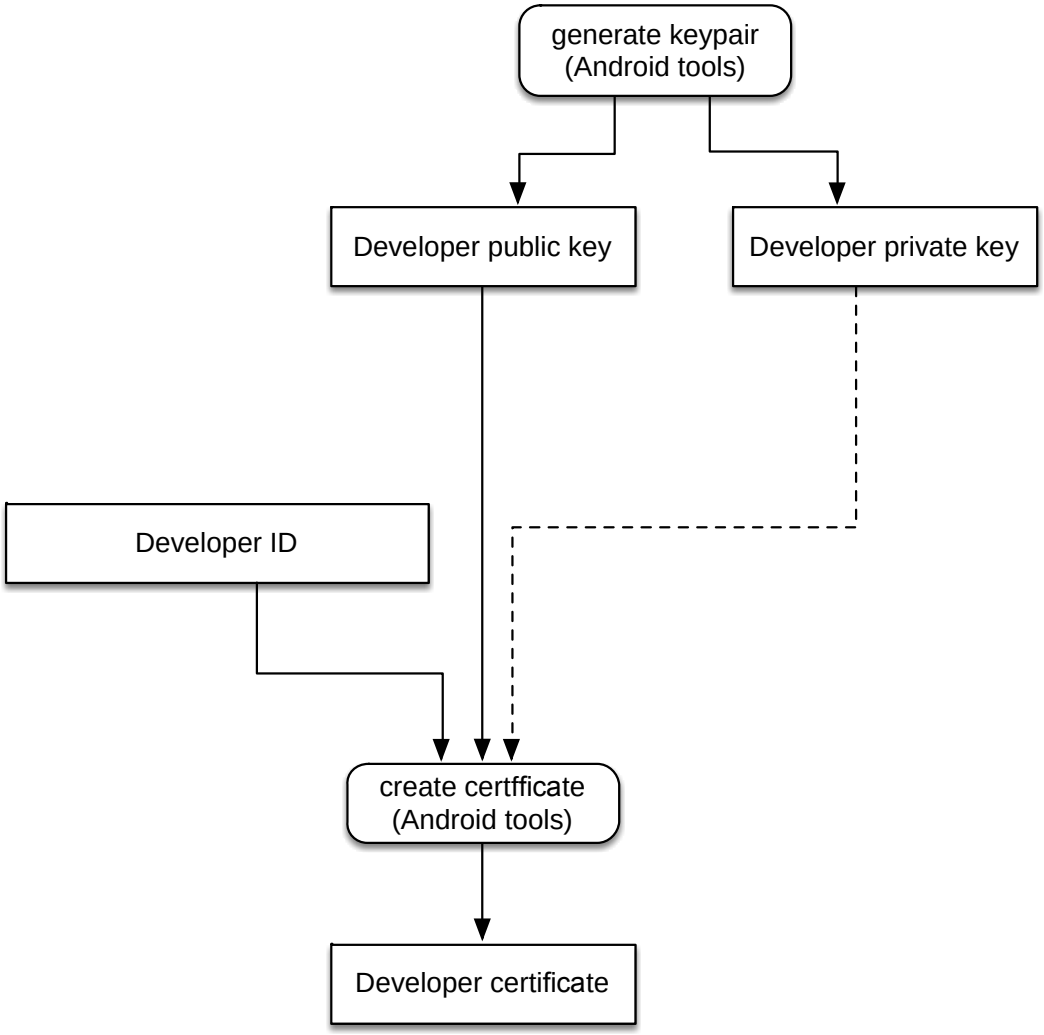

Figure 8： App developer creates release keys

5.3.4 Tag owner authorises 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.

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 Authorisation**.

The IARI Authorisation is an XML document in a GSMA-owned namespace that contains a **detached XML Digital Signature** 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. The detail of the IARI Authorisation document format is provided in section 7.

This step must be performed by the tag owner, and they can use GSMA-provided RCS tools. Once the IARI Authorisation document exists, it can be passed to the Developer and may then be used to sign multiple application packages without further reliance on the Tag owner.

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

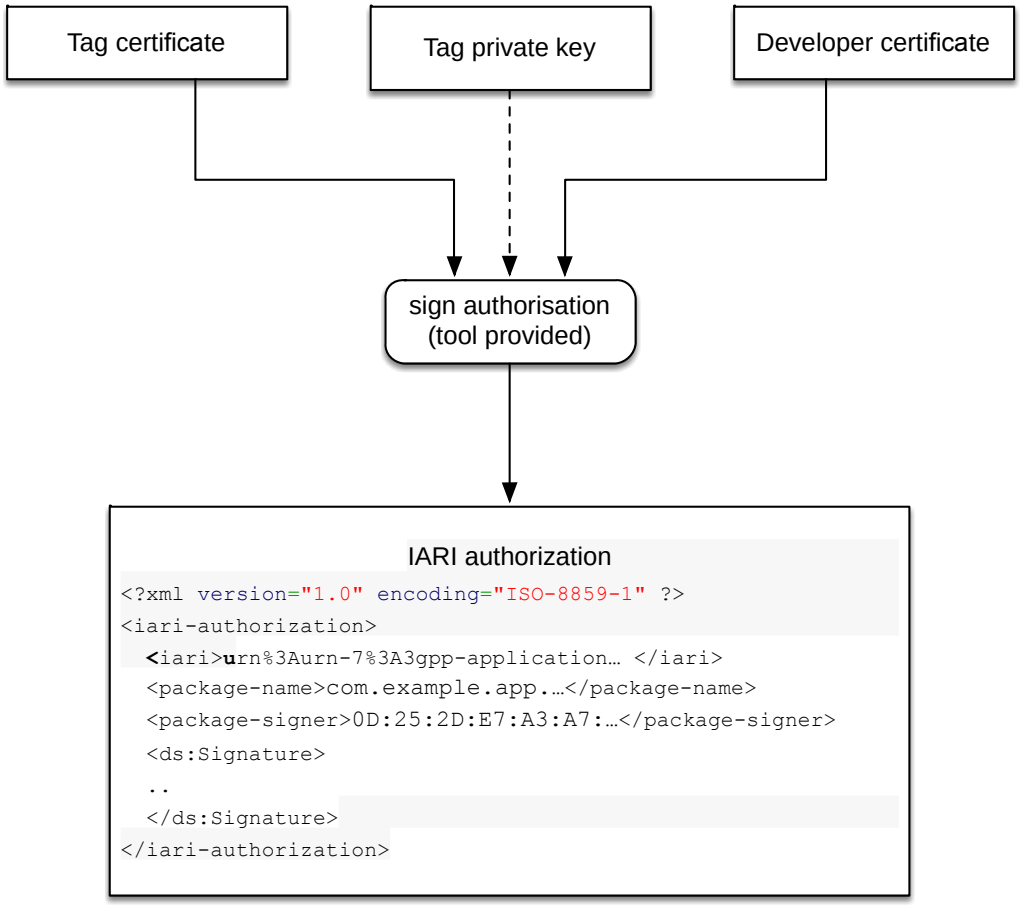

Figure 9: Tag owner authorises developer to use tag

5.3.5 Application developer creates and releases an app using the tag

By including the IARI Authorisation 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. The tag may also need to be used explicitly in calls to the RCS API, for example when initiating a private app-to-app session using that tag.

Now the application developer must add the IARI Authorisation to his application package. In the case of an Android application package, the IARI Authorisation 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 Authorisation document is added, the developer signs and releases his app in the usual way. The steps are illustrated in the figure below.

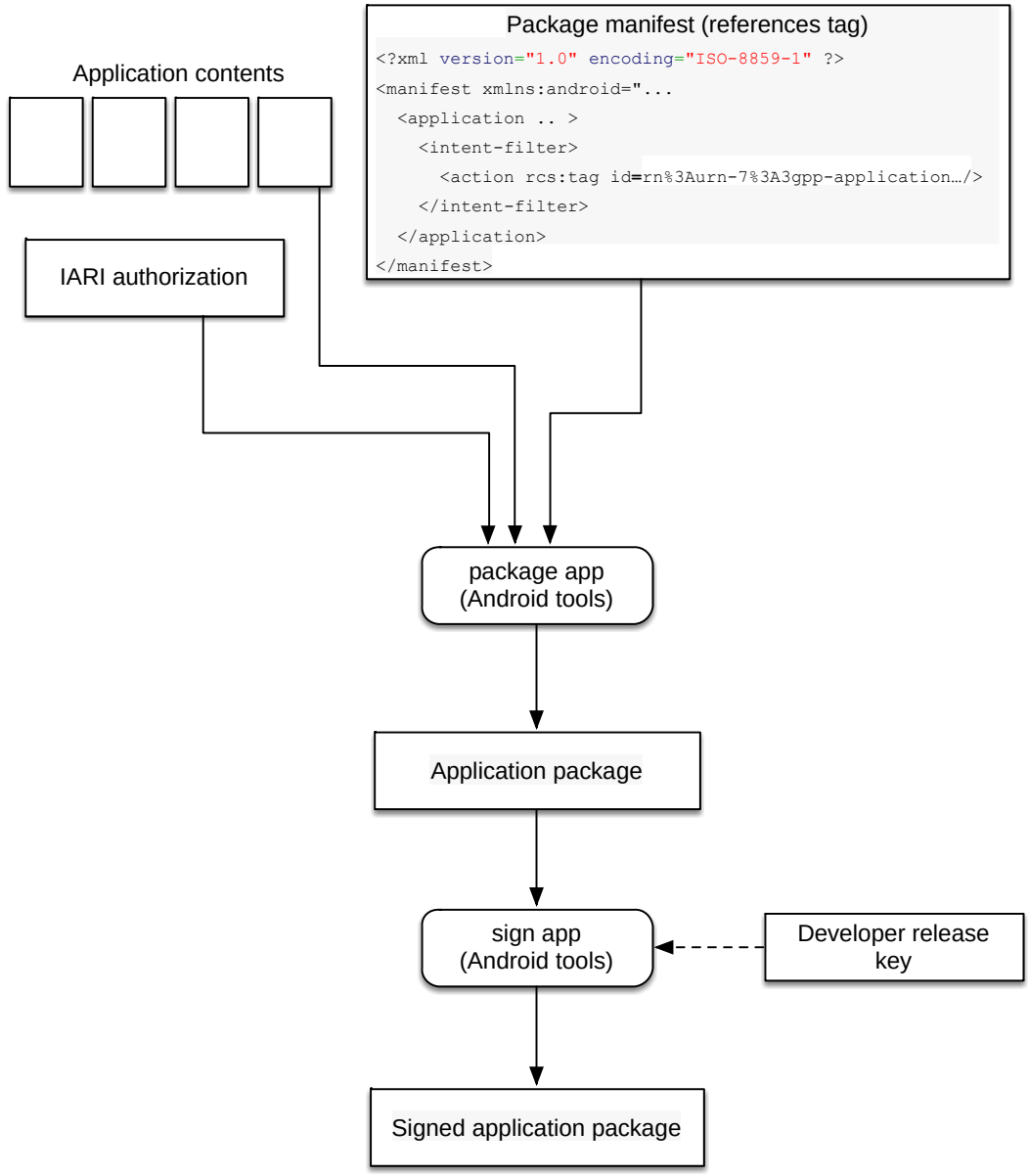

Figure 10: App developer creates and releases app using tag

5.3.6 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 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 package signature and ID correspond to IARI Authorisation.** For each Tag authorization present, the stack must verify that the authorization 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 authorization with the package signer certificate available from the package manager.
3. **Validate IARI Authorisation.** This step consists of the validation of the IARI Authorisation 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 Authorisation 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 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 below.

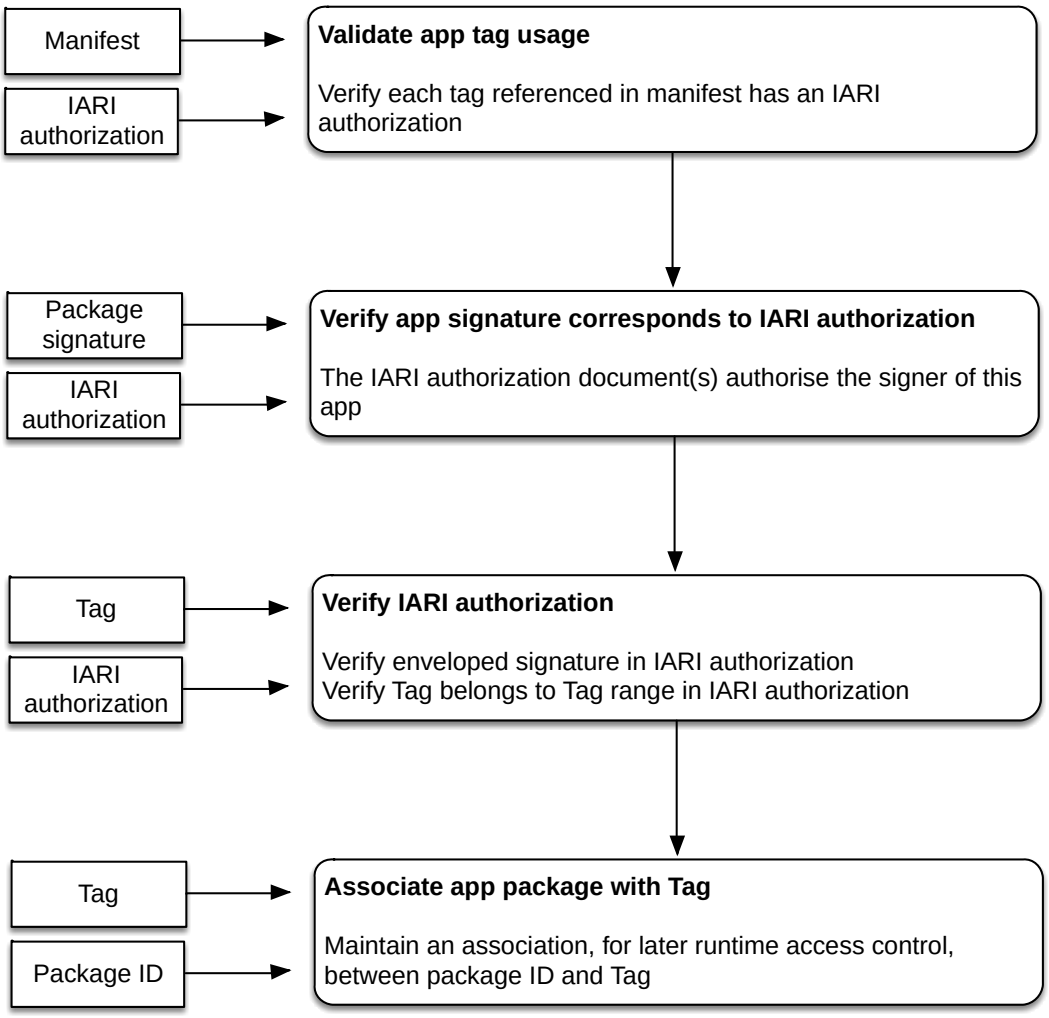

Figure 11: Stack validates app

5.3.7 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 authorised. 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 authorization to use the tag based on the previously determined association.

This is illustrated below.

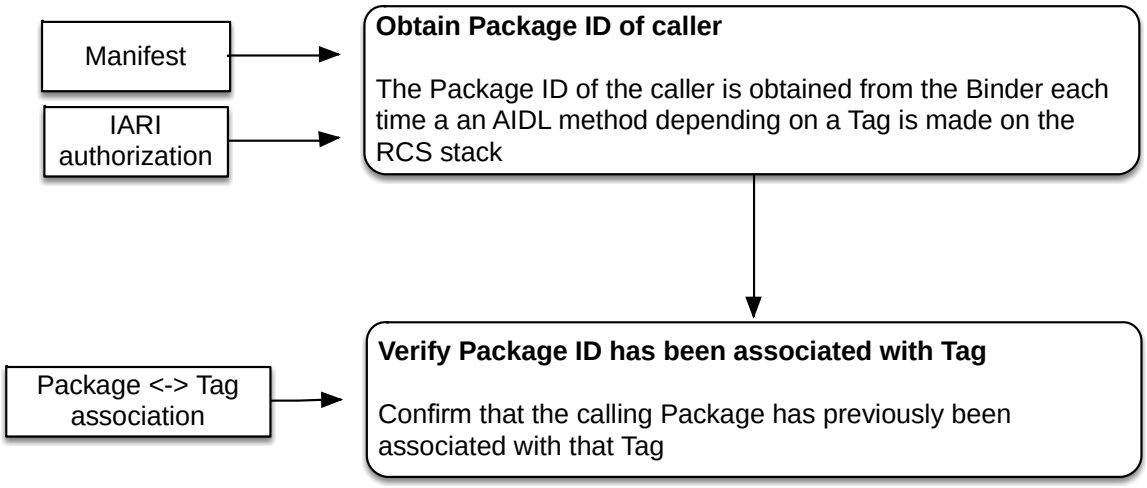

Figure 12: Stack validates runtime invocation

5.3.8 Tag validity: assurances provided to the user and developer

A valid stack will ensure that only an app with a valid IARIAuthorisation can listen for, and initiate, sessions, and all such sessions are appropriately labelled with that an IARI associated with the app. Non-conforming stacks, or stacks running on rooted devices, cannot provide those assurances.

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 authorized 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 akin to opening a port on an internet-connected device; when there is a connection to that port, the listening application must implement measures commensurate 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.

5.3.9 Tag registration and developer identification

As described above in section 3.6, it is expected there would be a method for developer/ application registration.

Based on such a developer/application database, it would also be possible to expose that tag registry in searchable form. The information in such a database is not seen as being a runtime dependency of the RCS stack or the service, but may be useful in tracking apps and their distribution. The 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.

5.3.10 Tag pre-registration

A service provider (MNO) might have a requirement for more specific control over third-party apps, requiring explicit approval before granting any access to controlled RCS 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.

The actual approvals/verifications/blocking processes implemented by any MNO are subject to individual organisational requirements.

The 'federated' model as described in section 3.5, agreed for RCS services enables MNOs to share information about developers and applications with their peers.

5.3.11 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 for specific applications (tags).

Since there is no central issuing authority, it is not possible to use a revocation mechanism based on the tag. 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 considered when the RCS traffic is being handled.

A mechanism is provided also, in the case of T-APIs, 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, with each MNO expected to hold a cached copy, 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 (see section 3.12.4.3 of [1]). 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.

6 Network API

6.1 Introduction

This section sets out the proposed approach to access control for the Network APIs, including the mechanisms proposed for independent authorisation of apps.

6.2 Network API application identification and access control

Network APIs will use HTTPS (over TCP/IP) as a bearer for the requests, API calls use OMA defined RESTful interfaces, e.g. for chat messaging.

In the OMA RESTful interfaces, the expectation is that applications are identified and authorised either:

- In the form of a client ID and optional client secret which are combined into an HTTP Basic Authorisation header (see IETF RFC 2617 - https://www.ietf.org/rfc/rfc2617.txt), or,
- In the form of an OAuth 2 "Access Token" (see http://tools.ietf.org/html/rfc6749) which has been provided to the application via an external authorisation/ consent process. The OAuth2 Authorisation process utilises the client ID and optional client secret again in the form of an HTTP Authorisation header which includes the "Access Token" granted to the application as a "Bearer Token" to authorise access to the RCS APIs. The access token is obtained through processes that leverage HTTP Basic Authorisation (RFC 2617).

Further details on OMA application identification/authorisation is defined within the OMA Autho4API Specification (http://technical.openmobilealliance.org//Technical/Release_Program/docs/Autho4API/V1_0-20131120-C/OMA-ER-Autho4API-V1_0-20131120-C.pdf).

In either case, therefore, the Network API gateway can first authorise and secondly, identify the application via the client ID associated with the application.

Any Network API gateway receiving an API call must be able to uniquely identify the client application, this creates a requirement that client IDs must be unique as far as any API gateway is concerned. Two approaches are possible:

- Developers obtain a client ID for their application from each operator they wish to work with, the operator ensures uniqueness of client ID but only on their own network. The developer application is then responsible for selecting the appropriate client ID for the Network API gateway it is working with;
- Developers obtain a client ID from a single registration platform which ensures global uniqueness of client IDs.

The format and length of a client ID is determined by the issuer and the application treats this simply as an opaque value which is used as part of the authorisation header for N-API requests.

A IARI Authorisation document is generated that binds a specific app identified with its client ID(s) to that IARI; 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 detail of the IARI Authorisation document format for N-API is provided in Section 7 and 8 of this document. As an extension to the OMA Network API mechanisms the application must add a custom HTTP header to the Network API Request 'X-RCS-IARI' to identify the routing between applications – whether the 'B party' is using a Network API or Terminal API based application. It is expected that the Network API gateway will use the ‘Authorisation Gateway’ and API Validator' services at the MNO to perform the relevant checks on the application including verification that the IARI matches with the IARI authorisation document which references the application's client ID.

The Network API gateway will ensure that the HTTP X-RCS-IARI header is converted as needed to/ from SIP Contact and Accept-Contact headers as required.

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 (MNO).

6.3 Network API application identification management

6.3.1 Overview

The following application identification management has been decided for Network API applications:

- Client IDs are issued to Network API applications as described above normally as part of application registration,
- The process for generating IARI (application identifiers) to be used for Network API applications follows exactly the same process as is described for Terminal API applications in Sections 5.3.2.,
- The tag owner will follow the processes as described in section 5.3.4 above to create a tag authorisation document – the document including the application Client ID (or IDs) so that the signature validates the association of IARI and Client ID's,

- The developer uploads the tag authorisation document to the central developer/ application database. MNOs then apply the information pushed to them within their API Validator' services along with any other factors relevant to approving the RCS API request e.g.

    - Global application (IARI) blocking,
    - Local application (IARI) blocking,
    - Developer acceptance of MNO Terms & Conditions,
    - Application approval.
    - Applications invokes N-API

Please refer to Section 3.5 for implementation models.

6.3.2 Client ID generation

As described in section 6.2, a client ID and secret is created for the app for Network API request authorisation generally by the developer registering the app on a developer portal. This step may be repeated in case the developer must register the app separately for each operator they are working with (i.e. each independent operator, or each entry operator for each operator federation). Below information is stored in the operator's repository to support Request Validation:

- Developer information (name, contact details, company name, terms and conditions acceptance, etc.),

- Application information (name, category, etc.).

6.3.3 Tag owner creates tag

Same process exists for T-APIs, refer to section 5.3.2.

6.3.4 Tag owner authorises developer to use tag

The IARI Authorisation is created and signed by the tag owner (as described in Section 5.3.5 above), this includes the IARI and the client ID (or IDs) for the app.

6.3.5 Application developer releases an app using the tag

If the MNO is participating in an RCS API federation, as defined in section 3.5:

a) The IARI Authorisation is 'uploaded' to the federation as part of the application provisioning process, the application is provisioned into the Central Developer/ Application Registry. The central platform should at this time validate the IARI Authorisation is valid for the app/client ID,

b) The central platform distributes the registered developer/ application/ IARI authorisation document to each operator's Local Cache Repository to support Request Validation.

6.3.6 Developer/ Application Approval

A service provider (MNO) might have a requirement for more specific control over third-party apps, requiring explicit approval before granting any access to controlled RCS Network 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 for operators to implement a check in the network based on the IARI presented in a session when an application invokes a Network API. Such a check should be initiated in the RCS N-API authorisation gateway which is expected to delegate most functions to the API Validator' function supporting T-API and N-API applications. Since the check is online, and the check is made using a request validation service provided by or for the MNO, the provider has ultimate control over whether or not any given access request is permitted.

6.3.7 RCS network validates runtime invocation

The application will prepare an N-API request (e.g. chat, file transfer) as follows:

1. The application will determine the relevant N-API endpoints and client ID/ secret to use. This might be based on an external discovery service (which is not part of the scope of this document);
2. The application will form an HTTP Authorisation header as described in section 8. This is added to the N-API request;
3. The application will add a custom HTTP header ('X-RCS-IARI') to the N-API request specifying the IARI which is being used for the 'B' party traffic;
4. The application will then make the respective OMA RCS API call.

When the API request is received at the Network API gateway the following process is expected to ensure a Network API based application has permitted access to an IARI.

6. Extract the client ID and optional client secret from the HTTP Authorisation header of the Network API request. If a "Bearer Token" is provided in the Authorisation header resolve the original client ID used when the token was issued;
7. Extract the IARI associated with the Network API request, from the custom HTTP header 'X-RCS-IARI';
8. Check the 'Local Cache Repository' that the client ID is known to be associated with a provisioned, active (i.e. not deleted by the developer) application and the client secret (if specified) matches the client ID;
9. Check the application is 'approved' for use on the network (approval might be automatic or manual);
10. Check that the developer organisation who registered the application is marked as having agreed the current (or relevant) Terms & Conditions for the operator;
11. Check that there is a (validated) IARI Authorisation document uploaded or referenced by the X-RCS-IARI Authorisation header against the application which includes the IARI and client ID of the Network API request;
12. Check that the IARI is not blocked at the global level (Local Cache Repository);
13. Check that the IARI is not blocked at the local operator level.

The Operator's RCS N-API Gateway would conduct steps 1 and 2 of this process.

The Authorisation Gateway (Client ID Authorisation function, as defined in section 3.5) would conduct steps 3 to 5, relying on Operator's API Validator.

The Authorisation Gateway (IARI Authorisation function, as defined in section 3.5) would conduct steps 6 and 7, relying on Operator's API Validator.

6.3.8 Tag blocking

Even with application approval processes, 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 for specific application tags.

Individual operators are always able to block tags locally via a bespoke blacklist/whitelist mechanism which would normally be a part of the API Validator function for the MNO. The centralised database also offers a means of blocking tags 'network wide' for the case that the abusive application is operating across multiple networks of an operators federation.

Centralised blocking should be under the authority of the 'Developer Operator' who registered the developer/application to the centralised database. In this case the application is blocked centrally and updates sent to all operator's local cache of the developer/ application database marking the application as blocked.

It is also envisaged that tag blocking can be used as a temporary measure in some cases, e.g. whilst getting a developer to implement a new version of their application which is compliant with requirements. Therefore, there is also required to be an equivalent tag un-blocking mechanism.

7 IARI Authorisation Document Specification

7.1 Introduction

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

7.2 Standalone Authorisation Document

This section covers the format and processing of a *standalone IARI Authorisation* which captures authorisation of an app to use a self-signed IARI.

The document format is extensible in principle to support authorisation for other IARI types but this is beyond the scope of the current specification.

7.3 Namespace

The IARI Authorisation namespace URI for an IARI Authorisation document is:

http://gsma.com/ns/iari-authorisation#

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-authorisation` element

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

Context in which this element is used	The `iari-authorisation` element is the root element of the IARI Authorisation document.
Occurrences	Exactly one, at the root element of the XML document.
Expected children	`iari`: 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 Authorisation document applies.

Context in which this element is used	In the `iari-authorisation` 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 `package-name` element

The `package-name` element represents the T-API application package identifier to which this IARI Authorisation document applies. A given `package-name` value matches an Android application package if it matches the value of the `package` attribute of the `manifest` element of the application manifest.

Context in which this element is used	In the `iari-authorisation` element.
Content model	A string value, equalling the value of the `package` attribute in the `<manifest>` of an Android application.
Occurrences	Zero or one for IARI Authorisation documents applying to an Android T-API application package.
Expected children	None
Attributes	`Id`: optional, type ID.

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

7.7 `package-signer` element

The `package-signer` element represents the T-API application package signer to which this IARI Authorisation 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-authorisation` element.
Content model	A string containing the SHA1 fingerprint of the package signature entity certificate.
Occurrences	Exactly one for IARI Authorisation documents applying to an Android T-API application package.
Expected children	None
Attributes	`Id`: optional, type ID.

7.8 `client_id` element

The `client_id` element represents an N-API Client Identifier to which this IARI Authorisation document applies. A given `client_id` value matches an N-API client application if it is equal to the Client Identifier associated with the Authorisation Grant used for N-API access.

Context in which this element is used	In the `iari-authorisation` element.
Content model	A string value, equalling the Client Identifier associated with the Authorisation Grant used for N-API access.
Occurrences	At least one for IARI Authorisation documents applying to an N-API client application.
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 Authorisation 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 authorisation) or provably belongs to the IARI owner (in the case of a standalone IARI Authorisation).

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-authorisation` 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.
Occurrences	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 ishttp://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 Authorisation, 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 CRLand/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-authorisation-role-standalone.

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-authorisation-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 Authorisation document processing

An RCS stack or the 'API Validator' function invoked by an authorisation gateway processes an IARI Authorisation 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 Authorisation document with an XML parser that is namespace-aware. If the document is not a well-formed XML then the processor must terminate these steps and treat the IARI Authorisation as invalid.
2. If the document element is not an `iari-authorisation` element in the `iari-authorisation` namespace then the processor must terminate these steps and treat the IARI Authorisation 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 `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,
c) 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,
d) If the element is a `client_id` element, let *client-id* be the text content of this element. If the element is not the first `client_id` 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* has not been assigned after processing all of the elements, the processor must terminate these steps and treat the IARI Authorisation as invalid.
5. If exactly one of *package-signer* or *client-id* has not been assigned after processing all of the elements, the processor must terminate these steps and treat the IARI Authorisation as invalid.
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 Authorisation as invalid,
b) Check that signature has a `ds:Reference` for each of the `iari`, `range`, `package-name`, `package-signature` and `client_id` elements present. If any such element exists without a reference, then the processor must terminate these steps and treat the IARI Authorisation 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 the processor must terminate these steps and treat the IARI Authorisation 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 Authorisation 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 Authorisation as invalid,
g) 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 Authorisation as invalid,
h) Optionally, validate any other SignatureProperties supported by the processor in the manner specified in [Signature Properties],
i) Perform reference validation and signature validation on the signature. If validation fails, then the processor must terminate these steps and treat the IARI Authorisation as invalid.

7. Check that the IARI Authorisation 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.

8. If *package-signer* is set, check that the T-API application package associated with the IARI Authorisation 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*.

9. If *client-id* is set, verify that the N-API client application associated with the IARI Authorisation matches:

   a) Verify that *client-id* matches the Client Identifier associated with the Authorisation Grant presented when attempting access to the N-API.

10. If the IARI Authorisation is valid according to the above steps, then the stack or the Request Validator invoked by the N-API gateway may permit the application to use iari.

8 Use of IARI Authorisation in N-API

8.1 Introduction

This section defines the headers and error indications associated with the use of IARI Authorisation documents in N-API accesses.

8.2 IARI Authorisation in N-API API access

The GSMA RCS N-API provides access to RCS over an REST-style HTTP API. This section defines how IARI Authorisations are referenced when accessing the N-API in the REST binding. No other bindings of the N-API are considered.

8.2.1 X-RCS-IARI

This header must be included in N-API requests for all service invocations associated with a custom (i.e. second-party or third-party) IARI. It is mandatory to include this header for any request to the following methods:

The following example shows how the X-RCS-IARI header should appear in an HTTP request generated using the curl utility:

```
$ curl -v http://gsma.com --basic -u "clientid:clientsecret" -H "X-RCS-
IAIR:ABC...DEF"
* Connected to gsma.com (46.137.85.46) port 80 (#0)
```

```
* Server auth using Basic with user 'clientid'
GET / HTTP/1.1
Authorisation: Basic Y2xpZW50aWQ6Y2xpZW50c2VjcmV0
User-Agent: curl/7.33.0
Host: gsma.com
Accept: */*
X-RCS-IAIR:ABC...DEF
```

NOTE: This example also shows how the clientid (which is synonymous with username in HTTP Basic Auth) and client secret (synonymous with password) is combined and base64 encoded as specified in RFC 2617 (http://tools.ietf.org/html/rfc2617).

The header value is the IARI URI string encoded using the application/x-www-form-urlencoded format (“URL-encoded”) defined in RFC6749 Appendix B.

It is not valid to reference multiple IARIs in a N-API request.

8.2.2 X-RCS-IARIAuthorisation

This header may be included in any N-API request that also has an X-RCS-IARI header. It references an IARI Authorisation document that provides authorisation for the request.

The reference must be a valid HTTP or HTTPS URL from which the IARI Authorisation request may be obtained via GET.

The header value is the URL-encoded URI location of the IARI Authorisation document.

For example, the use of the X-RCS-IARIAuthorization header in the HTTP request would be as follows:

```
HEAD / HTTP/1.1
Authorisation: Basic Y2xpZW50SWQ6Y2xpZW50U2VjcmV0
Host: gsma.com
Accept: application/json
X-RCS-IARIAuthorization:
http%3A%2F%2Fgsma.com%2FRCS%2FIARIAuthorizationExample.xml
```

It is not valid to reference multiple IARIs authorisation documents in a N-API request.

NOTE: That if this field is omitted, the assumption is that the document has been uploaded into the federated central developer/application repository and is, therefore, available to the MNO 'Request Validator' service via their local cache of the developer/application repository.

Network API requests which specify an IARI but have no accompanying authorisation document available should be rejected as 'Non Authorised' as below.

8.3 Error responses associated with IARI authorisation in N-API

A range error conditions may arise in connection with processing an N-API request involving an IARIAuthorisation. The specific error conditions and their associated error codes are defined below.

Error responses will follow the OMA defined formats which combines a 4xx or 5xx series status code and a JSON based error message

```
HTTP/1.1 400  Bad Request
Content-Type: application/json
Content-Length: 1234
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"requestError": {
    "serviceException": {
        "messageId": "SVC0002",
        "text": " Invalid input value for message part %1",
        "variables": " tel:+016309700000"
    }
}}
```

400 errors:

- Missing or invalid IARI reference.
- Unknown IARI.

401 errors:

- Missing or invalid IARIAuthorisation reference.
- Invalid IARIAuthorisation document
- Inapplicable IARIAuthorisation document
- IARIAuthorisation expired
- IARIAuthorisation revoked

403 errors:

- IARI forbidden for API access
- IARI blocked for API access

9 Device Provisioning for API

9.1 Introduction

The configuration parameters available in RCS 5.3 [1] will be extended with following additional provisioning parameter which controls the API access control policy of the MNO by standardised means.

9.2 Device Management parameters for API policy

Configuration parameter	Description	Notes
EXTENSIONS POLICY	This parameter indicates to a device whether or not non-native applications are permitted to access the T-API If this parameter is set to: 0, Only natively integrated Extensions are authorised to access and use the RCS service via the T-API. Non native applications are not permitted to access the T-API. 1, Natively-integrated Extensions and self-signed Extensions are authorised to access the RCS service via the T-API. For self-signed Extensions, the app accessing the API shall have an IARI Authorisation 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

Table 1: RCS API Extensions Policy configuration parameters

The Extensions Policy is placed in APIExt MO sub tree, located in the Ext node of the other subtree defined in section A.2.10 of RCS 5.3 [1].

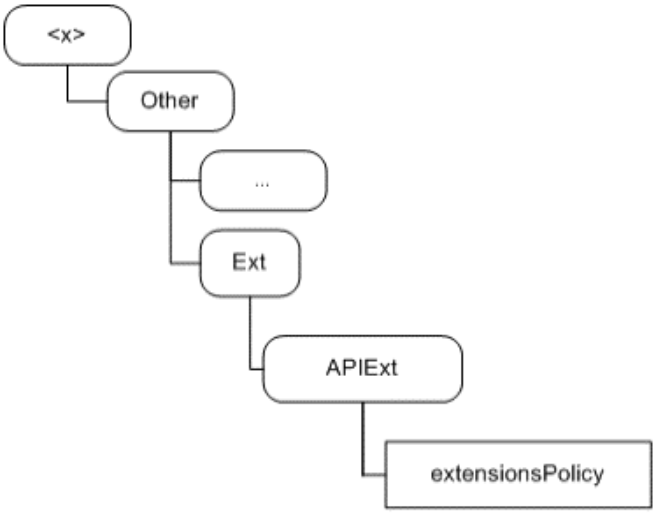

Figure 1: New APIExt sub tree in Other Ext MO

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

```
<characteristic type=" APIExt">
  <parm name="extensionsPolicy" value="X"/>
</characteristic>
```

Table 2: APIExt sub tree associated HTTP configuration XML structure

This structure will be included into the configuration document defined in section A.3 of RCS 5.3 [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>
                </characteristic>
            </characteristic>
            <characteristic type="SERVICEPROVIDEREXT">
                  …
            </characteristic>
        </characteristic>
</wap-provisioningdoc>
```

Table 3: 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 4: Other MO sub tree APIExt node

- Values: N/A
- Type property of the node is: urn:gsma:mo:rcs-other:5.3:Ext:APIExt
- Associated HTTP XML characteristic type: "APIEXT"

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

Leaf node that describes the types of Extensions authorised 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 5: APIExt MO sub tree addition parameters (extensionsPolicy)

- Values:

  0 - Only natively integrated Extensions are authorised to access and use the RCS service via the T-API.
  1 – Natively-integrated Extensions and self-signed Extensions are authorised to access the RCS service via the T-API. For self-signed Extensions, the app accessing the API shall have an IARI Authorisation 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 5.3 [1].
- Associated HTTP XML parameter ID: “extensionsPolicy”

Annex A Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	15 October 2014	Initial version	RCS APICOM PSMC	P. Byers, D. O'Byrne, Kelvin Qin / GSMA
2.0	23 July 2015	Security Framework and Network API integrations, tag registration and application blocking subjects are added.	RCS APICOM PSMC	P. Byers, Stephen Doyle, Erdem Ersoz / GSMA

A.2 Other Information

Type	Description
Document Owner	RCS APICOM
Editor / Company	Paddy Byers, Stephen Doyle, Erdem Ersoz / 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.