Network Working Group
Request for Comments: 5627
Category: Standards Track

J. Rosenberg
Cisco Systems

October 2009

Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)

Abstract

Several applications of the Session Initiation Protocol (SIP) require a user agent (UA) to construct and distribute a URI that can be used by anyone on the Internet to route a call to that specific UA instance. A URI that routes to a specific UA instance is called a Globally Routable UA URI (GRUU). This document describes an extension to SIP for obtaining a GRUU from a registrar and for communicating a GRUU to a peer within a dialog.

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Table of Contents

1. Introduction

In the Session Initiation Protocol (SIP), RFC 3261 [1], the basic unit of reference is the Address of Record (AOR). However, in SIP systems a single user can have a number of user agents (handsets, softphones, voicemail accounts, etc.) that are all referenced by the same AOR. There are a number of contexts in which it is desirable to have an identifier that addresses a single user agent rather than the group of user agents indicated by an AOR.

As an example, consider a blind transfer application (see RFC 5589 [19]). User A is talking to user B. User A wants to transfer the call to user C. So, user A sends a REFER to user C. That REFER looks like, in part:

```
REFER sip:C@example.com SIP/2.0
From: sip:A@example.com;tag=99asd
To: sip:C@example.com
Refer-To: (URI that identifies B's UA)
```

The Refer-To header field needs to contain a URI that can be used by user C to place a call to user B. However, this call needs to route to the specific UA instance that user B is using to talk to user A. If it doesn't, the transfer service will not execute properly. For example, if A provides C with B's AOR, the call might be routed to B's voicemail rather than B's current handset.

In order to enable this functionality, user B provides an instance-specific URI to user A in the Contact header of their SIP exchange. This URI refers to the user agent B is currently using, and it can be dereferenced by C's user agent. Because user B doesn't know in advance who user A will transfer the call to, the URI has to be usable by anyone.

Many current clients attempt to meet the need for an instance-specific identifier by using explicit IP addresses in the values they provide in the Contact header field. However, this interacts poorly with NATs and firewalls, and as a practical matter, these URIs cannot be used by arbitrary external clients. Usage of hostnames has proven problematic for similar reasons. In addition, many SIP clients do not have or cannot obtain a hostname for themselves at all.

This specification describes a mechanism for providing a unique user-agent identifier which is still globally routable. This identifier is called a Globally Routable User Agent (UA) URI (GRUU).

2. Terminology

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 RFC 2119 [4].

This specification defines the following additional terms:

contact: The term "contact", when used in all lowercase, refers to a URI that is bound to an AOR and GRUU by means of a registration. A contact is usually a SIP URI, and is bound to the AOR and GRUU through a REGISTER request by appearing as a value of the Contact header field. The contact URI identifies a specific UA.

remote target: The term "remote target" refers to a URI that a user agent uses to identify itself for receipt of both mid-dialog and out-of-dialog requests. A remote target is established by placing a URI in the Contact header field of a dialog-forming request or response and is updated by target refresh requests or responses.

Contact header field: The term "Contact header field", with a capitalized C, refers to the header field that can appear in REGISTER requests and responses, redirects, or dialog-creating requests and responses. Depending on the semantics, the Contact header field sometimes conveys a contact, and sometimes conveys a remote target.

3. Overview of Operation

The basic idea behind a GRUU is simple. GRUUs are issued by SIP domains and always route back to a proxy in that domain. In turn, the domain maintains the binding between the GRUU and the particular UA instance. When a GRUU is dereferenced while sending a SIP request, that request arrives at the proxy. It maps the GRUU to the contact for the particular UA instance, and sends the request there.

3.1. Structure of GRUUs

A GRUU is a SIP URI that has two properties:

- It routes to a specific UA instance.
- It can be successfully dereferenced by any user agent on the Internet, not just ones in the same domain or IP network as the UA instance to which the GRUU points.

In principle, a GRUU can be constructed in any way the domain chooses, as long as it meets the criteria above. However, all GRUUs contain the "gr" URI parameter (either with or without a value), so that a recipient of a GRUU can tell that it has these two properties.

In practice, there are two different types of GRUUs:

1. GRUUs that expose the underlying AOR

2. GRUUs that hide the underlying AOR

3.1.1. GRUUs That Expose the Underlying AOR

In many cases, it is desirable to construct the GRUU in such a way that the mapping to the AOR is apparent. For example, many user agents retain call logs, which keep track of incoming and outgoing call attempts. If the UA had made a call to a GRUU (perhaps as a consequence of a transfer request), the call log will contain the GRUU. Since the call log is rendered to the user, it would be useful to be able to present the user with the AOR instead, since the AOR is meaningful to users as an identifier.

This type of GRUU is called a public GRUU. It is constructed by taking the AOR, and adding the "gr" URI parameter with a value chosen by the registrar in the domain. The value of the "gr" URI parameter contains a representation of the UA instance. For instance, if the AOR was "sip:alice@example.com", the GRUU might be:

```
sip:alice@example.com;gr=kjh29x97us97d
```

If a UA removes the "gr" URI parameter, the result is the AOR. Since many systems ignore unknown parameters anyway, a public GRUU will "look" like the AOR to those systems.

3.1.2. GRUUs That Hide the Underlying AOR

In other cases, it is desirable to construct a GRUU that obfuscates the AOR such that it cannot be extracted by a recipient of the GRUU. Such a GRUU is called a temporary GRUU. The most obvious reason to do this is to protect the user’s privacy. In such cases, the GRUU can have any content, provided that it meets the requirements in Sections 3.1 and 5.4, and the AOR cannot be readily determined from the GRUU. The GRUU will have the "gr" URI parameter, either with or without a value. In order to avoid creating excessive state in the registrar, it is often desirable to construct cryptographically protected "stateless" GRUUs using an algorithm like that described in Appendix A.

An example of a temporary GRUU constructed using a stateful algorithm would be:

```
sip:asd887f9dfkk76690@example.com;gr
```

3.2. Obtaining a GRUU

A user agent can obtain a GRUU in one of several ways:

- As part of its REGISTER transaction.
- By constructing one locally, using the IP address or hostname of the user agent instance as the domain part of the URI. These are called self-made GRUUs, and are only really GRUUs when constructed by UAs that know they are globally reachable using their IP address or hostname.
- Via some locally specified administrative mechanism.

A UA that wants to obtain a GRUU via its REGISTER request does so by providing an instance ID in the "+sip.instance" Contact header field parameter, defined in RFC 5626 [14]. For example:

```
Contact: <sip:callee@192.0.2.2>
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
```

The registrar detects this header field parameter and provides two GRUUs in the REGISTER response. One of these is a temporary GRUU, and the other is the public GRUU. These two GRUUs are returned in the "temp-gruu" and "pub-gruu" Contact header field parameters in the response, respectively. For example:

```
<allOneLine>
Contact: <sip:callee@192.0.2.2>
;pub-gruu="sip:callee@example.com;gr=urn:
uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;temp-gruu="sip:tgruu.7hs==
jd7vnzga5w7fajsc7-ajd6fabz0f8g5@example.com;gr"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=3600
</allOneLine>
```

Note that the <allOneLine> tag is used as defined in [17].

When a user agent refreshes this registration prior to its expiration, the registrar will return back the same public GRUU, but will create a new temporary GRUU. Despite the fact that each refresh provides the UA with a new temporary GRUU, all of the temporary GRUUs

learned from previous REGISTER responses during the lifetime of a contact remain valid as long as (1) a contact with that instance ID remains registered, and (2) the UA doesn't change the Call-ID in its REGISTER request compared to previous ones for the same reg-id [14]. When the last contact for the instance expires, either through explicit de-registration or timeout, all of the temporary GRUUs become invalidated. Similarly, if a register refresh for a contact (or, if RFC 5626 is being used, for a reg-id) changes the Call-ID compared to previous register refreshes, all of the previous temporary GRUUs are invalidated. When the user agent later creates a new registration with the same instance ID, the public GRUU is the same. The temporary GRUU will be new (as it is with refreshes), and it will be the only valid temporary GRUU for the instance until the next refresh, at which point a second one becomes valid too. Consequently, temporary GRUUs "accumulate" during the lifetime of a registration.

3.3. Using a GRUU

Once a user agent obtains GRUUs from the registrar, it uses them in several ways. First, it uses them as the contents of the Contact header field in non-REGISTER requests and responses that it emits (for example, an INVITE request and 200 OK response). According to RFC 3261 [1], the Contact header field is supposed to contain a URI that routes to that user agent. Prior to this specification, there hasn't been a way to really meet that requirement. The user agent would use one of its temporary GRUUs for anonymous calls, and use its public GRUU otherwise.

Second, the UA can use the GRUU in any other place it needs to use a URI that resolves to itself, such as a webpage.

3.4. Dereferencing a GRUU

Because a GRUU is simply a URI, a UA dereferences it in exactly the same way as it would any other URI. However, once the request has been routed to the appropriate proxy, the behavior is slightly different. The proxy will map the GRUU to the AOR and determine the set of contacts that the particular UA instance has registered. The GRUU is then mapped to those contacts, and the request is routed towards the UA.

4. User Agent Behavior

This section defines the normative behavior for user agents.

4.1. Generating a REGISTER Request

When a UA compliant to this specification generates a REGISTER request (initial or refresh), it MUST include the Supported header field in the request. The value of that header field MUST include "gruu" as one of the option tags. This alerts the registrar for the domain that the UA supports the GRUU mechanism.

Furthermore, for each contact for which the UA desires to obtain a GRUU, the UA MUST include a "sip.instance" media feature tag (see RFC 5626 [14]) as a UA characteristic (see [7]), whose value MUST be the instance ID that identifies the UA instance being registered. Each such Contact header field SHOULD NOT contain a "pub-gruu" or "temp-gruu" header field. The contact URI MUST NOT be equivalent, based on the URI equality rules in RFC 3261 [1], to the AOR in the To header field. If the contact URI is a GRUU, it MUST NOT be a GRUU for the AOR in the To header field.

As in RFC 3261 [1], the Call-ID in a REGISTER refresh SHOULD be identical to the Call-ID used to previously register a contact. With GRUU, an additional consideration applies. If the Call-ID changes in a register refresh, the server will invalidate all temporary GRUUs associated with that UA instance; the only valid one will be the new one returned in that REGISTER response. When RFC 5626 is in use, this rule applies to the reg-ids: If the Call-ID changes for the registration refresh for a particular reg-id, the server will invalidate all temporary GRUUs associated with that UA instance as a whole. Consequently, if a UA wishes its previously obtained temporary GRUUs to remain valid, it MUST utilize the same Call-ID in REGISTER refreshes. However, it MAY change the Call-ID in a refresh if invalidation is the desired objective.

Note that, if any dialogs are in progress that utilize a temporary GRUU as a remote target, and a UA performs a registration refresh with a change in Call-ID, those temporary GRUUs become invalid, and the UA will not be reachable for subsequent mid-dialog messages.

If a UA instance is trying to register multiple contacts for the same instance for the purposes of redundancy, it MUST use the procedures defined in RFC 5626 [14].

> A UA utilizing GRUUs can still perform third-party registrations and can include contacts that omit the "+sip.instance" Contact header field parameter.

If a UA wishes to guarantee that the REGISTER request is not processed unless the domain supports and uses this extension, it MAY include a Require header field in the request with a value that contains the "gruu" option tag. This is in addition to the presence of the Supported header field, also containing the "gruu" option tag. The use of Proxy-Require is not necessary and is NOT RECOMMENDED.

4.2. Learning GRUUs from REGISTER Responses

If the REGISTER response is a 2xx, each Contact header field that contains the "+sip.instance" Contact header field parameter can also contain a "pub-gruu" and "temp-gruu" Contact header field parameter. These header field parameters convey the public and a temporary GRUU for the UA instance, respectively. A UA MUST be prepared for a Contact header field to contain just a "pub-gruu", just a "temp-gruu", neither, or both. The temporary GRUU will be valid for the duration of the registration (that is, through refreshes), while the public GRUU persists across registrations. The UA will receive a new temporary GRUU in each successful REGISTER response, while the public GRUU will typically be the same. However, a UA MUST be prepared for the public GRUU to change from a previous one, since the persistence property is not guaranteed with complete certainty. If a UA changed its Call-ID in this REGISTER request compared to a previous REGISTER request for the same contact or reg-id, the UA MUST discard all temporary GRUUs learned through prior REGISTER responses. A UA MAY retain zero, one, some, or all of the temporary GRUUs that it is provided during the time over which at least one contact or reg-id remains continuously registered. If a UA stores any temporary GRUUs for use during its registration, it needs to be certain that the registration does not accidentally lapse due to clock skew between the UA and registrar. Consequently, the UA MUST refresh its registration such that the REGISTER refresh transaction will either complete or timeout prior to the expiration of the registration. For default transaction timers, this would be at least 32 seconds prior to expiration, assuming the registration expiration is larger than 64 seconds. If the registration expiration is less than 64 seconds, the UA SHOULD refresh its registration halfway prior to expiration.

Note that, when [14] is in use, and the UA is utilizing multiple flows for purposes of redundancy, the temporary GRUUs remain valid as long as at least one flow is registered. Thus, even if the registration of one flow expires, the temporary GRUUs learned previously remain valid.

In cases where registrars forcefully shorten registration intervals, the registration event package, RFC 3680 [24], is used by user agents to learn of these changes. A user agent implementing both RFC 3680 [24] and GRUU MUST also implement the extensions to RFC 3680 [24] for

conveying information on GRUU, as defined in RFC 5628 [28], as these are necessary to keep the set of temporary GRUUs synchronized between the UA and the registrar. More generally, the utility of temporary GRUUs depends on the UA and registrar being in sync on the set of valid temporary GRUUs at any time. Without support of RFC 3680 [24] and its extension for GRUU, the client will remain in sync only as long as it always re-registers well before the registration expiration. Besides forceful de-registrations, other events (such as network outages, connection failures, and short refresh intervals) can lead to potential inconsistencies in the set of valid temporary GRUUs. For this reason, it is RECOMMENDED that a UA that utilizes temporary GRUUs implement RFC 3680 [24] and RFC 5628 [28].

A non-2xx response to the REGISTER request has no impact on any existing GRUUs previously provided to the UA. Specifically, if a previously successful REGISTER request provided the UA with a GRUU, a subsequent failed request does not remove, delete, or otherwise invalidate the GRUU.

The user and host parts of the GRUU learned by the UA in the REGISTER response MUST be treated opaquely by the UA. That is, the UA MUST NOT modify them in any way. A UA MUST NOT modify or remove URI parameters it does not recognize. Furthermore, the UA MUST NOT add, remove, or modify URI parameters relevant for receipt and processing of request at the proxy, including the transport, lr, maddr, ttl, user, and comp (see RFC 3486 [25]) URI parameters. The other URI parameter defined in RFC 3261 [1], method, would not typically be present in a GRUU delivered from a registrar, and a UA MAY add a method URI parameter to the GRUU before handing it out to another entity. Similarly, the URI parameters defined in RFC 4240 [26] and RFC 4458 [27] are meant for consumption by the UA. These would not be included in the GRUU returned by a registrar and MAY be added by a UA wishing to provide services associated with those URI parameters.

Note, however, that should another UA dereference the GRUU, the parameters will be lost at the proxy when the Request-URI is translated into the registered contact, unless some other means is provided for the attributes to be delivered to the UA. Mechanisms for such delivery are currently the subject of future standardization activity (see "Delivery of Request-URI Targets to User Agents" [29]).

4.3. Constructing a Self-Made GRUU

Many user agents (such as gateways to the Public Switched Telephone Network (PSTN), conferencing servers, and media servers) do not perform registrations, and cannot obtain GRUUs through that mechanism. These types of user agents can be publicly reachable. This would mean that the policy of the domain is that requests can

come from anywhere on the public Internet and be delivered to the user agent without requiring processing by intervening proxies within the domain. Furthermore, firewall and NAT policies administered by the domain would allow such requests into the network. When a user agent is certain that these conditions are met, a UA MAY construct a self-made GRUU. Of course, a user agent that does REGISTER, but for whom these conditions are met regardless, MAY also construct a self-made GRUU. However, usage of GRUUs obtained by the registrar is RECOMMENDED instead.

A self-made GRUU is one whose domain part equals the IP address or hostname of the user agent. The user part of the SIP URI is chosen arbitrarily by the user agent. Like all other GRUUs, the URI MUST contain the "gr" URI parameter, with or without a value, indicating it is a GRUU.

If a user agent does not register, but is not publicly reachable, it would need to obtain a GRUU through some other means. Typically, the UA would be configured with a GRUU, the GRUU would be configured into the proxy, and the proxy will be configured with a mapping from the GRUU to the IP address (or hostname) and port of the UA.

4.4. Using One's Own GRUUs

A UA SHOULD use a GRUU when populating the Contact header field of dialog-forming and target refresh requests and responses. In other words, a UA compliant to this specification SHOULD use one of its GRUUs as its remote target. This includes:

- the INVITE request
- a 2xx or 18x response to an INVITE which contains a To tag
- the SUBSCRIBE request (see [5])
- a 2xx response to a SUBSCRIBE which contains a To tag
- the NOTIFY request
- the REFER request (see [6])
- a 2xx response to NOTIFY
- the UPDATE request
- a 2xx response to NOTIFY

The only reason not to use a GRUU would be privacy considerations; see Section 10.3.

When using a GRUU obtained through registrations, a UA MUST have an active registration prior to using a GRUU, and MUST use a GRUU learned through that registration. It MUST NOT reuse a GRUU learned through a previous registration that has lapsed (in other words, one obtained when registering a contact that has expired). The UA MAY use either the public or one of its temporary GRUUs provided by its registrar. A UA MUST NOT use a temporary GRUU learned in a REGISTER response whose Call-ID differs from the one in the most recent REGISTER request generated by the UA for the same AOR and instance ID (and, if RFC 5626 [14] is in use, reg-id). When a UA wishes to construct an anonymous request as described in RFC 3323 [15], it SHOULD use a temporary GRUU. See Section 10.3 for a more complete discussion on the level of privacy afforded by temporary GRUUs.

As per RFC 3261 [1], a UA SHOULD include a Supported header with the option tag "gruu" in requests and responses it generates.

4.4.1. Considerations for Multiple AORs

In some SIP networks, a user agent can have a multiplicity of AORs, either in different domains or within the same domain. In such cases, additional considerations apply.

When a UA sends a request, the request will be sent 'using' one of its AORs. This AOR will typically show up in the From header field of the request, and credentials unique to that AOR will be used to authenticate the request. The GRUU placed into the Contact header field of such a request SHOULD be one that is associated with the AOR used to send the request. In cases where the UA uses a tel URI (as defined in [11]) to populate the From header field, the UA typically has a SIP AOR that is treated as an alias for the tel URI. The GRUU associated with that SIP AOR SHOULD be used in the Contact header field.

When a UA receives a request, the GRUU placed into the Contact header field of a 2xx response SHOULD be the one associated with the AOR or GRUU to which the request was most recently targeted. There are several ways to determine the AOR or GRUU to which a request was sent. For example, if a UA registered a different contact to each AOR (by using a different user part of the URI), the Request-URI (which contains that contact) will indicate the AOR.

4.5. Dereferencing a GRUU

A GRUU is identified by the presence of the "gr" URI parameter, and this URI parameter might or might not have a value. A UA that wishes to send a request to a URI that contains a GRUU knows that the request will be delivered to a specific UA instance without further action on the part of the requestor.

Some UAs implement non-standard URI-handling mechanisms that compensate for the fact that heretofore many contact URIs have not been globally routable. Since any URI containing the "gr" URI parameter is known to be globally routable, a UA SHOULD NOT apply such mechanisms when a contact URI contains the "gr" URI parameter.

> Because the instance ID is a callee capabilities parameter, a UA might be tempted to send a request to the AOR of a user, and include an Accept-Contact header field (defined in [12]) that indicates a preference for routing the request to a UA with a specific instance ID. Although this would appear to have the same effect as sending a request to the GRUU, it does not. The caller preferences expressed in the Accept-Contact header field are just preferences. Their efficacy depends on a UA constructing an Accept-Contact header field that interacts with domain-processing logic for an AOR, to cause a request to route to a particular instance. Given the variability in routing logic in a domain (for example, time-based routing to only selected contacts), this doesn't work for many domain-routing policies. However, this specification does not forbid a client from attempting such a request, as there can be cases where the desired operation truly is a preferential routing request.

4.6. Rendering GRUUs on a User Interface

When rendering a GRUU to a user through a user interface, it is RECOMMENDED that the "gr" URI parameter be removed. For public GRUUs, this will produce the AOR, as desired. For temporary GRUUs, the resulting URI will be seemingly random. Future work might provide improved mechanisms that would allow an automaton to know that a URI is anonymized, and therefore inappropriate to render.

5. Registrar Behavior

5.1. Processing a REGISTER Request

A REGISTER request might contain a Require header field with the "gruu" option tag; this indicates that the registrar has to understand this extension in order to process the request. It does not require the registrar to create GRUUs, however.

As the registrar is processing the contacts in the REGISTER request according to the procedures of step 7 in Section 10.3 of RFC 3261 [1], the registrar checks whether each Contact header field in the REGISTER message contains a "+sip.instance" header field parameter. If present with a non-zero expiration, the contact is processed further based on the rules in the remainder of this section. Otherwise, the contact is processed based on normal RFC 3261 [1] rules.

> Note that handling of a REGISTER request containing a Contact header field with value "*" and an expiration of zero still retains the meaning defined in RFC 3261 [1] -- all contacts, not just those with a specific instance ID, are deleted. As described in Section 5.4, this removes the binding of each contact to the AOR and the binding of each contact to its GRUUs.

If the contact URI is equivalent (based on URI equivalence in RFC 3261 [1]) to the AOR, the registrar MUST reject the request with a 403, since this would cause a routing loop. If the contact URI is a GRUU for the AOR in the To header field of the REGISTER request, the registrar MUST reject the request with a 403, for the same reason. If the contact is not a SIP URI, the REGISTER request MUST be rejected with a 403.

Next, the registrar checks if there is already a valid public GRUU for the AOR (present in the To header field of the REGISTER request) and the instance ID (present as the content of the "+sip.instance" Contact header field parameter). If there is no valid public GRUU, the registrar SHOULD construct a public GRUU at this time according to the procedures of Section 5.4. The public GRUU MUST be constructed by adding the "gr" URI parameter, with a value, to the AOR. If the contact contained a "pub-gruu" Contact header field parameter, the header field parameter MUST be ignored by the registrar. A UA cannot suggest or otherwise provide a public GRUU to the registrar.

Next, the registrar checks for any existing contacts registered to the same AOR, instance ID, and if the contact in the REGISTER request is registering a flow [14], reg-id. If there is at least one, the registrar finds the one that was most recently registered, and examines the Call-ID value associated with that registered contact. If it differs from the one in the REGISTER request, the registrar MUST invalidate all previously generated temporary GRUUs for the AOR and instance ID. A consequence of this invalidation is that requests addressed to those GRUUs will be rejected by the domain with a 404 from this point forward.

Next, the registrar SHOULD create a new temporary GRUU for the AOR and instance ID with the characteristics described in Section 5.4. The temporary GRUU construction algorithm MUST have the following two properties:

1. The likelihood that the temporary GRUU is equal to another GRUU that the registrar has created MUST be vanishingly small.

2. Given a pair of GRUUs, it MUST be computationally infeasible to determine whether they were issued for the same AOR or instance ID or for different AORs and instance IDs.

If the contact contained a "temp-gruu" Contact header field parameter, the header field parameter MUST be ignored by the registrar. A UA cannot suggest or otherwise provide a temporary GRUU to the registrar.

5.2. Generating a REGISTER Response

When generating the 200 (OK) response to the REGISTER request, the procedures of step 8 of Section 10.3 of RFC 3261 [1] are followed. Furthermore, for each Contact header field value placed in the response, if the registrar has stored an instance ID associated with that contact, that instance ID is returned as a Contact header field parameter. If the REGISTER request contained a Supported header field that included the "gruu" option tag, and the registrar has at least one temporary GRUU assigned to the instance ID and AOR, the registrar MUST add a "temp-gruu" Contact header field parameter to that Contact header field. The value of the "temp-gruu" parameter is a quoted string, and MUST contain the most recently created temporary GRUU for that AOR and instance ID. In addition, if the registrar has a public GRUU assigned to the instance ID and AOR (and the client supports GRUUs), the registrar MUST add a "pub-gruu" Contact header field parameter to that Contact header field. The value of the "pub-gruu" Contact header field parameter is the public GRUU.

The registrar SHOULD NOT include the "gruu" option tag in the Require or Supported header field of the response.

5.3. Timing Out a Registration

When a registered contact expires (either due to timeout or explicit de-registration), its binding to the AOR is removed as usual. In addition, its binding to its GRUUs are removed at the same time, as a consequence of the relationships described in Section 5.4

If, as a consequence of the expiration of the contact, a particular GRUU no longer has any registered contacts bound to it, and the GRUU is a temporary GRUU, the GRUU MUST be invalidated. This means that all of the accumulated temporary GRUUs get invalidated once the last contact for a given instance ID expires.

If, however, the GRUU was a public GRUU, the registrar SHOULD continue to treat the GRUU as valid. Consequently, subsequent requests targeted to the GRUU, prior to re-registration of a contact to the GRUU, SHOULD return a 480 (Temporarily Unavailable) response. In addition, since the GRUU remains valid, the rules in Section 5.1 will cause it to be retained when a contact with that instance ID is once again registered to the AOR.

> These rules give a public GRUU a semi-permanent property. The intent is that the registrar make every attempt to retain validity of the GRUU for as long as the AOR itself is known within the domain. The requirements for doing so are at SHOULD strength and not MUST strength because of the difficulty in meeting a MUST strength requirement; registrar failures could cause the set of valid GRUUs to be lost, and this specification requires the UA to be robust against such cases. That said, it is possible for a public GRUU to be constructed such that a registrar does not need to retain any additional state for it, yet the GRUU still meets the requirements described here.

5.4. Creation of a GRUU

This section defines additional behaviors associated with the construction and maintenance of a GRUU that are specific to a registrar. These rules do not apply to self-made GRUUs or GRUUs not obtained through registrations.

When a registrar creates a GRUU, it is required to maintain certain information associated with the GRUU, regardless of whether it is a public or temporary GRUU. Every GRUU is associated with a single AOR and a single instance ID. A registrar MUST be able to determine the instance ID and AOR when presented with a GRUU. In addition, the GRUU, like an AOR, resolves to zero or more contacts. While the AOR resolves to all registered contacts for an AOR, a GRUU resolves only to those contacts whose instance ID matches the one associated with the GRUU. For this reason, a contact with an instance ID is always bound to both a GRUU and its AOR, never just an AOR or just a GRUU. This is shown pictorially in Figure 1. The figure shows three contacts registered to a single AOR. One of the contacts has an instance ID of 1, and the other two have an instance ID of 2. There are two GRUUs for this AOR. One is associated with instance ID 1, and the other with instance ID 2. The first GRUU resolves only to

contacts whose instance ID is 1, and the second resolves only to contacts whose instance ID is 2. There will typically be multiple contacts for a given instance ID if a UA has crashed, rebooted, and re-registered with the same instance ID, or is using the mechanisms of RFC 5626 [14] to have multiple registrations for redundancy. If the contact for instance ID 1 expires, the AOR would resolve to two contacts, but the GRUU associated with instance ID 1 would resolve to zero.

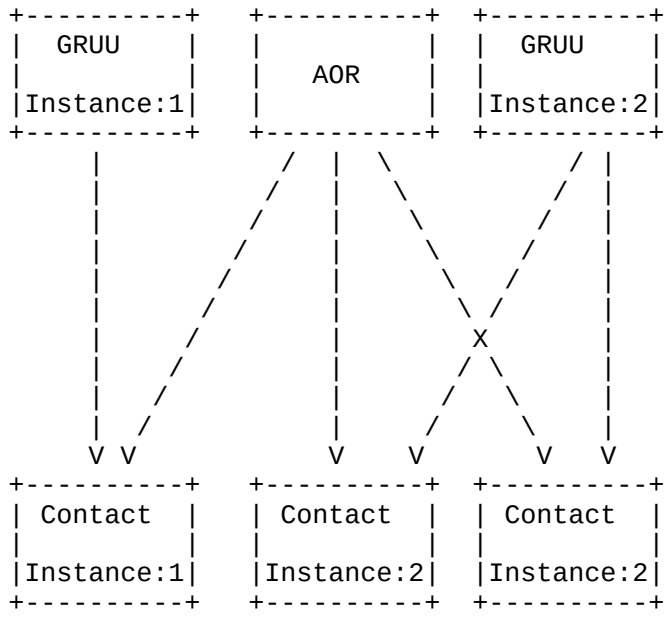

Figure 1

There can be multiple GRUUs with the same instance ID and AOR. Indeed, this specification requires registrars to maintain many -- one that is public, and several that are temporary. However, if two GRUUs are associated with different AORs or different instance IDs or both, the GRUUs MUST be different based on URI equality comparison. A GRUU in a domain MUST NOT be equivalent, based on URI comparison, to any AOR in a domain except for the one associated with the GRUU.

A public GRUU will always be equivalent to the AOR based on URI equality rules. The reason is that the rules in RFC 3261 [1] cause URI parameters that are in one URI, but not in the other, to be ignored for equality purposes. Since a public GRUU differs from an AOR only by the presence of the "gr" URI parameter, the two URIs are equivalent based on those rules.

Once a temporary GRUU is constructed, it MUST be considered valid by the registrar until invalidated based on the rules described previously. Once a public GRUU is constructed, it MUST be considered valid for the duration that the AOR itself is valid. Once an AOR is no longer valid within a domain, all of its GRUUs MUST be considered invalid as well.

This specification does not mandate a particular mechanism for construction of the GRUU. Example algorithms for public and temporary GRUUs that work well are given in Appendix A. However, in addition to the properties described in Section 3.1, a GRUU constructed by a registrar MUST exhibit the following properties:

- The domain part of the URI is an IP address present on the public Internet, or, if it is a hostname, the resolution procedures of RFC 3263 [2], once applied, result in an IP address on the public Internet.

- When a request is sent to the GRUU, it routes to a proxy that can access the registration data generated by the registrar. Such a proxy is called an authoritative proxy, defined in RFC 5626 [14].

5.5. Registration Event Support

RFC 3680 [24] defines an event package that allows a client to learn about registration events at the registrar. This package allows registrars to alter registrations forcefully (for example, shortening them to force a re-registration). If a registrar is supporting RFC 3680 [24] and GRUU, it MUST also support RFC 5628 [28].

6. Proxy Behavior

Proxy behavior is fully defined in Section 16 of RFC 3261 [1]. GRUU processing impacts that processing in two places -- request targeting at the authoritative proxy and record-routing.

6.1. Request Targeting

When a proxy receives a request, owns the domain in the Request-URI, and is supposed to access a location service in order to compute request targets (as specified in Section 16.5 of RFC 3261 [1]), the proxy examines the Request-URI. If it contains the "gr" URI parameter but is not equivalent, based on URI comparison, to a currently valid GRUU within the domain, it SHOULD be rejected with a 404 (Not Found) response; this is the same behavior a proxy would exhibit for any other URI within the domain that is not valid.

If the Request-URI contains the "gr" URI parameter and is equivalent, based on URI comparison, to a GRUU which is currently valid within the domain, processing proceeds as it would for any other URI present in the location service, as defined in Section 16.5 of RFC 3261 [1], except that the "gr" URI parameter is not removed as part of the canonicalization process. This is the case for both out-of-dialog requests targeted to the GRUU, and mid-dialog requests targeted to the GRUU (in which case the incoming request would have a Route header field value containing the URI that the proxy used for record-routing.).

Note that the "gr" URI parameter is retained just for the purposes of finding the GRUU in the location service; if a match is found, the Request-URI will be rewritten with the registered contacts, replacing the GRUU and its "gr" URI parameter. The "gr" URI parameter is not carried forward into the rewritten Request-URI.

If there are no registered contacts bound to the GRUU, the server MUST return a 480 (Temporarily Unavailable) response. If there are more than one, there are two cases:

1. The client is using RFC 5626 [14] and registering multiple contacts for redundancy. In that case, these contacts contain "reg-id" Contact header field parameters, and the rules described in Section 7 of RFC 5626 [14] for selecting a single registered contact apply.

2. The client was not using RFC 5626 [14], in which case there would only be multiple contacts with the same instance ID if the client had rebooted, restarted, and re-registered. In this case, these contacts would not contain the "reg-id" Contact header field parameter. The proxy MUST select the most recently refreshed contact. As with RFC 5626, if a request to this target fails with a 408 (Request Timeout) or 430 (Flow Failed) response, the proxy SHOULD retry with the next most recently refreshed contact. Furthermore, if the request fails with any other response, the proxy MUST NOT retry on any other contacts for this instance.

Any caller preferences in the request (as defined in RFC 3841 [12]) SHOULD be processed against the contacts bound to the GRUU.

In essence, to select a registered contact, the GRUU is processed just like it was the AOR, but with only a subset of the contacts bound to the AOR.

Special considerations apply to the processing of any Path headers stored in the registration (see RFC 3327 [3]). If the received request has Route header field values beyond the one pointing to the

authoritative proxy itself (this will happen when the request is a mid-dialog request), the Path URI MUST be discarded. This is permitted by RFC 3327 [3] as a matter of local policy; usage of GRUUs will require this policy in order to avoid call spirals and likely call failures.

A proxy MAY apply other processing to the request, such as execution of called party features, as it might do for requests targeted to an AOR. For requests that are outside of a dialog, it is RECOMMENDED to apply screening types of functions, both automated (such as blacklist and whitelist screening) and interactive (such as interactive voice response (IVR) applications that confer with the user to determine whether to accept a call). In many cases, the new request is related to an existing dialog, and might be an attempt to join it (using the Join header field defined in RFC 3911 [21]) or replace it (using the Replaces header field defined in RFC 3891 [22]). When the new request is related to an existing dialog, the UA will typically make its own authorization decisions; bypassing screening services at the authoritative proxy might make sense, but needs to be carefully considered by network designers, as the ability to do so depends on the specific type of screening service.

However, forwarding services, such as call forwarding, SHOULD NOT be provided for requests sent to a GRUU. The intent of the GRUU is to target a specific UA instance, and this is incompatible with forwarding operations.

If the request is a mid-dialog request, a proxy SHOULD only apply services that are meaningful for mid-dialog requests, generally speaking. This excludes screening and forwarding functions.

In addition, a request sent to a GRUU SHOULD NOT be redirected. In many instances, a GRUU is used by a UA in order to assist in the traversal of NATs and firewalls, and a redirection might prevent such a case from working.

6.2. Record-Routing

There are two distinct requirements for record-routing -- in the originating domain and in the terminating domain. These requirements avoid unnecessary, and possibly problematic, spirals of requests.

If:

- an originating authoritative proxy receives a dialog-forming request,

- AND the Contact header field contains a GRUU in the domain of the proxy,
- AND that GRUU is a valid one in the domain of the proxy,
- AND that GRUU is associated with the AOR matching the authenticated identity of the requestor (assuming such authentication has been performed),
- AND the request contains Record-Route header fields,

then the authoritative proxy MUST record-route. If all of these conditions are true, except that the GRUU is associated with an AOR that did not match the authenticated identity of the requestor, it is RECOMMENDED that the proxy reject the request with a 403 (Forbidden) response.

If:

- a terminating authoritative proxy receives a dialog-forming request,
- AND the Request-URI contains a URI in the location service (either a GRUU or an AOR),
- AND the contact selected for sending the request has an instance ID and is bound to a GRUU,
- AND the registration contain Path URI,

then the authoritative proxy MUST record-route.

If a proxy is in either the originating or terminating domains but is not an authoritative proxy, the proxy MAY record-route.

If a proxy in the terminating domain requires mid-dialog requests to pass through it for whatever reason (firewall traversal, accounting, etc.), the proxy MUST still record-route, and MUST NOT assume that a UA will utilize its GRUU in the Contact header field of its response (which would cause mid-dialog requests to pass through the proxy without record-routing).

> Implementors should note that, if a UA uses a GRUU in its contact, and a proxy inserted itself into the Path header field of a registration, that proxy will be receiving mid-dialog requests regardless of whether it record-routes or not. The only distinction is what URI the proxy will see in the topmost Route

header field of mid-dialog requests. If the proxy record-routes, it will see that URI. If it does not, it will see the Path URI it inserted.

7. Grammar

This specification defines two new Contact header field parameters ("temp-gruu" and "pub-gruu") by extending the grammar for "contact-params" as defined in RFC 3261 [1]. It also defines a new SIP URI parameter ("gr") by extending the grammar for "uri-parameter" as defined in RFC 3261 [1]. The ABNF [13] is as follows:

```
contact-params  =/ temp-gruu / pub-gruu
temp-gruu       =  "temp-gruu" EQUAL quoted-string
pub-gruu        =  "pub-gruu" EQUAL quoted-string

uri-parameter   =/ gr-param
gr-param        = "gr" ["=" pvalue]   ; defined in RFC 3261
```

The quoted strings for temp-gruu and pub-gruu MUST contain a SIP URI. However, they are encoded like all other quoted strings and can therefore contain quoted-pair escapes when represented this way.

8. Requirements

This specification was created in order to meet the following requirements:

REQ 1: When a UA invokes a GRUU, it must cause the request to be routed to the specific UA instance to which the GRUU refers.

REQ 2: It must be possible for a GRUU to be invoked from anywhere on the Internet, and still cause the request to be routed appropriately. That is, a GRUU must not be restricted to use within a specific addressing realm.

REQ 3: It must be possible for a GRUU to be constructed without requiring the network to store additional state.

REQ 4: It must be possible for a UA to obtain a multiplicity of GRUUs that each route to that UA instance. For example, this is needed to support ad hoc conferencing where a UA instance needs a different URI for each conference it is hosting. NOTE: This requirement is not met by this specification, and is being addressed in a separate specification (currently, "Delivery of Request-URI Targets to User Agents" [29]).

REQ 5: When a UA receives a request sent to a GRUU, it must be possible for the UA to know the GRUU that was used to invoke the request. This is necessary as a consequence of REQ 4. NOTE: This requirement is not met by this specification, and is being addressed in a separate specification (currently, "Delivery of Request-URI Targets to User Agents" [29]).

REQ 6: It must be possible for a UA to add opaque content to a GRUU. This content is not interpreted or altered by the network, and is used only by the UA instance to whom the GRUU refers. This provides a basic cookie type of functionality, allowing a UA to build a GRUU with the state embedded. NOTE: This requirement is not met by this specification, and is being addressed in a separate specification (currently, "Delivery of Request-URI Targets to User Agents" [29]).

REQ 7: It must be possible for a proxy to execute services and features on behalf of a UA instance represented by a GRUU. As an example, if a user has call-blocking features, a proxy might want to apply those call-blocking features to calls made to the GRUU, in addition to calls made to the user's AOR.

REQ 8: It must be possible for a UA in a dialog to inform its peer of its GRUU, and for the peer to know that the URI represents a GRUU. This is needed for the conferencing and dialog reuse applications of GRUUs, where the URIs are transferred within a dialog.

REQ 9: When transferring a GRUU per REQ 8, it must be possible for the UA receiving the GRUU to be assured of its integrity and authenticity.

REQ 10: It must be possible for a server that is authoritative for a domain to construct a GRUU that routes to a UA instance bound to an AOR in that domain. In other words, the proxy can construct a GRUU, too. This is needed for the presence application.

9. Example Call Flow

The following call flow, shown in Figure 2, shows a basic registration and call setup, followed by a subscription directed to the GRUU. It then shows a failure of the callee, followed by a re-registration. The conventions of RFC 4475 [17] are used to describe the representation of long message lines.

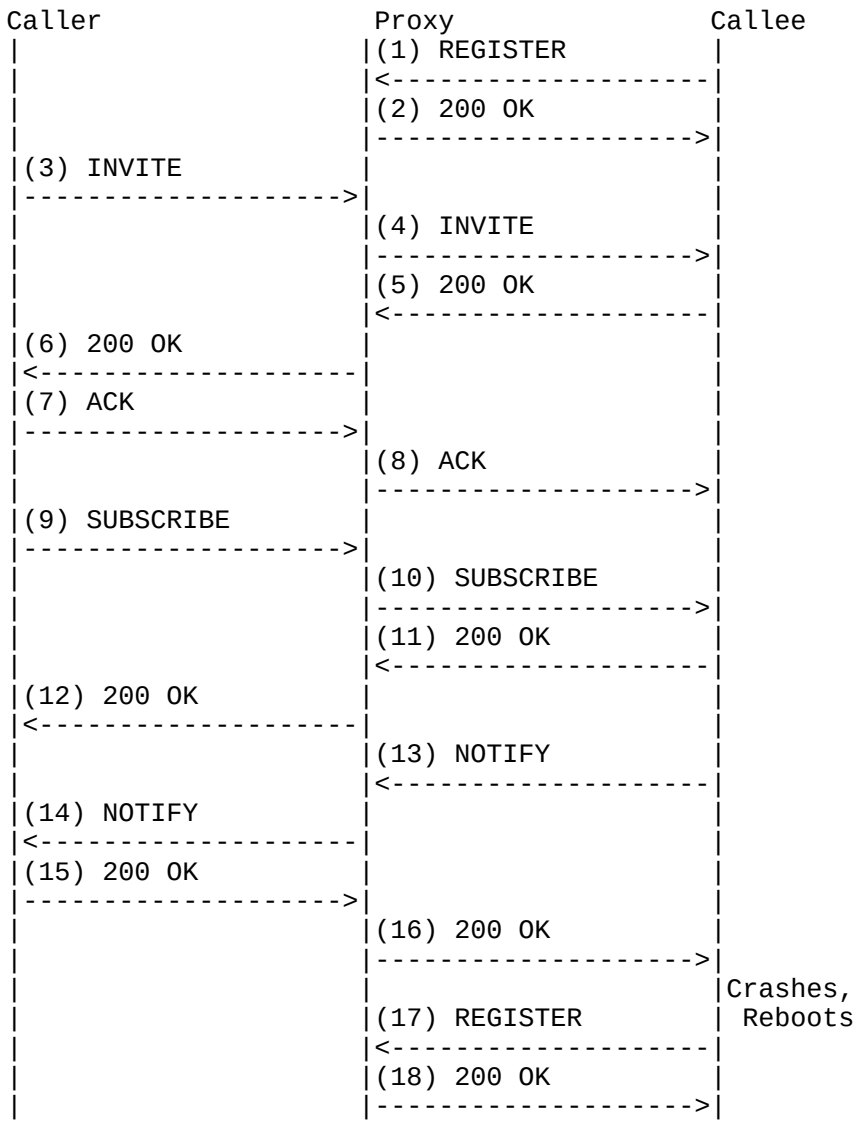
Caller
Proxy
Callee
(1) REGISTER
(2) 200 OK
(3) INVITE
(4) INVITE
(5) 200 OK
(6) 200 OK
(7) ACK
(8) ACK
(9) SUBSCRIBE
(10) SUBSCRIBE
(11) 200 OK
(12) 200 OK
(13) NOTIFY
(14) NOTIFY
(15) 200 OK
(16) 200 OK
Crashes,
Reboots
(17) REGISTER
(18) 200 OK

Figure 2

The callee supports the GRUU extension. As such, its REGISTER (1) looks like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
Supported: gruu
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn@192.0.2.1
CSeq: 1 REGISTER
Contact: <sip:callee@192.0.2.1>
 ;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
Content-Length: 0
```

The registrar assigns a temporary and a public GRUU. The REGISTER response (message 2) would look like:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com> ;tag=b88sn
Call-ID: 1j9FpLxk3uxtm8tn@192.0.2.1
CSeq: 1 REGISTER
<allOneLine>
Contact: <sip:callee@192.0.2.1>
;pub-gruu="sip:callee@example.com
;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;temp-gruu="sip:tgruu.7hs==
jd7vnzga5w7fajsc7-ajd6fabz0f8g5@example.com;gr"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=3600
</allOneLine>
Content-Length: 0
```

The Contact header field in the REGISTER response contains the "pub-gruu" Contact header field parameter with the public GRUU sip:callee@example.com;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6, and the "temp-gruu" header field parameter with the temporary GRUU sip:tgruu.7hs==jd7vnzga5w7fajsc7-ajd6fabz0f8g5@example.com;gr. Both are valid GRUUs for the AOR and instance ID, and both translate to the contact sip:callee@192.0.2.1.

The INVITE from the caller (message 3) is a normal SIP INVITE. However, the 200 OK generated by the callee (message 5) now contains a GRUU as the remote target. The UA has chosen to use its public GRUU.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bKnaa8
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK99a
From: Caller <sip:caller@example.com>;tag=n88ah
To: Callee <sip:callee@example.com> ;tag=a0z8
Call-ID: 1j9FpLxk3uxtma7@host.example.com
CSeq: 1 INVITE
Supported: gruu
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK, SUBSCRIBE
<allOneLine>
Contact:
<sip:callee@example.com
;gr=urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>
</allOneLine>
Content-Length: --
Content-Type: application/sdp

[SDP Not shown]
```

At some point later in the call, the caller decides to subscribe to the dialog event package (defined in [16]) at that specific UA. To do that, it generates a SUBSCRIBE request (message 9), but directs it towards the remote target, which is a GRUU:

```
<allOneLine>
SUBSCRIBE sip:callee@example.com;gr=urn:uuid:f8
1d4fae-7dec-11d0-a765-00a0c91e6bf6
 SIP/2.0
</allOneLine>
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:caller@example.com>;tag=kkaz-
<allOneLine>
To: <sip:callee@example.com;gr=urn:uuid:f8
1d4fae-7dec-11d0-a765-00a0c91e6bf6>
</allOneLine>
Call-ID: faif9a@host.example.com
CSeq: 2 SUBSCRIBE
Supported: gruu
Event: dialog
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK, NOTIFY
Contact: <sip:caller@example.com;gr=hdg7777ad7aflzig8sf7>
Content-Length: 0
```

In this example, the caller itself supports the GRUU extension and is using its own GRUU to populate its remote target.

This request is routed to the proxy, which proceeds to perform a location lookup on the Request-URI. It is translated into the contact for that instance, and then proxied to that contact.

```
SUBSCRIBE sip:callee@192.0.2.1 SIP/2.0
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bK9555
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:caller@example.com>;tag=kkaz-
<allOneLine>
To: <sip:callee@example.com;gr=urn:uuid:f8
1d4fae-7dec-11d0-a765-00a0c91e6bf6>
</allOneLine>
Call-ID: faif9a@host.example.com
CSeq: 2 SUBSCRIBE
Supported: gruu
Event: dialog
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK, NOTIFY
Contact: <sip:caller@example.com;gr=hdg7777ad7aflzig8sf7>
Content-Length: 0
```

The SUBSCRIBE generates a 200 response (message 11), which is followed by a NOTIFY (message 13 and 14) and its response (message 15 and 16). At some point after message 16 is received, the callee's machine crashes and recovers. It obtains a new IP address, 192.0.2.2. Unaware that it had previously had an active registration, it creates a new one (message 17 below). Notice how the instance ID remains the same, as it persists across reboot cycles:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.2;branch=z9hG4bKnasbba
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=ha8d777f0
Supported: gruu
To: Callee <sip:callee@example.com>
Call-ID: hf8asxzff8s7f@192.0.2.2
CSeq: 1 REGISTER
<allOneLine>
Contact: <sip:callee@192.0.2.2>
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
</allOneLine>
Content-Length: 0
```

The registrar notices that a different contact, sip:callee@192.0.2.1, is already associated with the same instance ID. It registers the new one too and returns both in the REGISTER response. Both have the same public GRUUs, but the registrar has generated a second temporary GRUU for this AOR and instance ID combination. Both contacts are

included in the REGISTER response, and the temporary GRUU for each is the same -- the most recently created one for the instance ID and AOR. The registrar then generates the following response:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.2;branch=z9hG4bKnasbba
From: Callee <sip:callee@example.com>;tag=ha8d777f0
To: Callee <sip:callee@example.com>;tag=99f8f7
Call-ID: hf8asxzff8s7f@192.0.2.2
CSeq: 1 REGISTER
<allOneLine>
Contact: <sip:callee@192.0.2.2>
;pub-gruu="sip:callee@example.com;gr=urn:
uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;temp-gruu="sip:tgruu.7hatz6cn-098shfyq193=
ajfux8fyg7ajqqe7@example.com;gr"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=3600
</allOneLine>
<allOneLine>
Contact: <sip:callee@192.0.2.1>
;pub-gruu="sip:callee@example.com;gr=urn:
uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
;temp-gruu="sip:tgruu.7hatz6cn-098shfyq193=
ajfux8fyg7ajqqe7@example.com;gr"
;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
;expires=400
</allOneLine>
Content-Length: 0
```

There is no need for the UA to remove the stale registered contact; the request targeting rules in Section 6.1 will cause the request to be delivered to the most recent one.

10. Security Considerations

Attacks in SIP networks using GRUUs can be divided into outside attacks (where a third party is trying to attack the system) and inside attacks (where the attacker is a valid participant in the system but is malicious). In addition, there are privacy considerations with using GRUUs.

10.1. Outside Attacks

It is important for a UA to be assured of the integrity of a GRUU given in a REGISTER response. If the GRUU is tampered with by an attacker, the result could be denial of service (DoS) to the UA. As a result, it is RECOMMENDED that a UA use the SIPS URI scheme in the

Request-URI when registering. Proxies and registrars MUST support the SIPS URI and MUST support TLS. This does not represent a change from the requirements in RFC 3261 [1].

The example GRUU construction algorithm in Appendix A.1 makes no attempt to create a GRUU that hides the AOR and instance ID associated with the GRUU. In general, determination of the AOR associated with a GRUU is considered a good property, since it allows for easy tracking of the target of a particular call. Learning the instance ID provides little benefit to an attacker. To register or otherwise impact registrations for the user, an attacker would need to obtain the credentials for the user. Knowing the instance ID is insufficient.

The example GRUU construction algorithm in Appendix A.1 makes no attempt to create a GRUU that prevents users from guessing a GRUU based on knowledge of the AOR and instance ID. A user that is able to do that will be able to direct a new request at a particular instance. However, this specification recommends that service treatment (in particular, screening features) be given to requests that are sent to a GRUU. That treatment will make sure that the GRUU does not provide a back door for attackers to contact a user that has tried to block the attacker.

10.2. Inside Attacks

As a consequence of this specification, a UA will begin using GRUUs in the dialog forming and target refresh requests and responses it emits. These GRUUs will be passed to another UA (called the correspondent), which then uses them in requests that they emit.

If a malicious correspondent removes the "gr" URI parameter, the request will be routed to the authoritative proxy. If the GRUU had been temporary, removal of the "gr" URI parameter produces a URI that is not recognized as a GRUU and is not equal to any AOR. The request will be rejected. If the GRUU had been public, removing the "gr" URI parameter would have produced the AOR. Therefore, the request is treated like a call to the AOR. Since it is a desired goal to allow users to extract the AOR from the GRUU, this is not an attack, and the call will be handled normally.

A malicious user in the system might try to use a GRUU for launching a DoS attack against another SIP UA. To do that, it would wait for a call from that UA, and from it, observe their GRUU. Once the GRUU is obtained, the UA would launch a SIP request to an entity, such as a presence server, which will generate many requests back towards the UA. However, the attacker will use the target’s GRUU in the Contact header field of that SUBSCRIBE request. This will cause the traffic

to be directed towards the target instead. Since the GRUU is globally routable, such traffic is more likely to be delivered to the target than traffic sent to its IP address. This specification helps mitigate this attack by requiring proxies to validate that the GRUU in the Contact of a request matches the authenticated identity of the sender of the request. This check requires the use of an outbound proxy. SIP does not require outbound proxies, and this does leave a potential area of vulnerability. However, in practice, nearly all deployments of SIP utilize an outbound proxy, and therefore this vulnerability is not likely to be a concern.

10.3. Privacy Considerations

RFC 3323 [15] defines mechanisms for privacy. It distinguishes between network-provided privacy and user-provided privacy. In the former, the user requests privacy services from the network by including a Privacy header field in the request. In the latter, the UA follows a basic set of guidelines for construction of its request, so a certain level of privacy is afforded.

The guidelines in Section 4.1 of RFC 3323 [15] for user-provided privacy request that a UA construct its Contact header field with a URI that omits a user part, and utilizes the IP address or hostname of the UA. Such recommendations are in conflict with the rules defined in this specification, which require the usage of a GRUU in the Contact header field.

However, the temporary GRUUs provided by the registrar can be used in place of the Contact URI format described in RFC 3323 [15]. A user agent would gather the temporary GRUU returned in each REGISTER response, and keep a small number of them cached. When it makes or receives a call, a temporary GRUU is used to populate the Contact header field.

A UA can either elect to use the same temporary GRUU in each call, or it can use a different temporary GRUU in each call. The choice depends on the level of privacy desired:

- A UA utilizing the same temporary GRUU for each call will allow a correspondent, based solely on investigation of the Contact header field, to correlate calls as coming from the same UA. This is also true for the user-provided privacy procedures in RFC 3323 [15], since the IP address or hostname in the Contact URI provides a similar correlator.

- A UA utilizing a different temporary GRUU for each call will not allow a correspondent, based solely on investigation of the Contact header field, to correlate calls as coming from the same UA.

- In both cases, absent network-provided privacy, IP address and port information in the Session Description Protocol (SDP) (defined in [10]) will allow a correspondent to correlate calls as coming from the same UA.

- In both cases, if a user makes a call, the correspondent will be able to call back by directing requests towards the GRUU in the Contact header field. Similarly, features such as transfer and digit collection by network application servers (see RFC 4730 [20]), which depend on a Contact with the GRUU property, will also be possible. These kinds of inbound requests will be possible until the registration for that UA lapses. A UA that wishes to invalidate its previous temporary GRUU in order to limit reachability MAY do so by generating a REGISTER refresh with a Call-ID that differs from ones used previously. A UA SHOULD NOT forcefully expire its registration and then re-register in order to invalidate a temporary GRUU; this results in a brief period of unreachability and will often produce excess load on the network. Refreshing with a new Call-ID is more efficient and is meant as the technique for coarse-grained control over the validity of temporary GRUUs. A UA wishing to not be disturbed by a specific call back will need to implement manual or automated call-handling procedures to reject it. This specification does not provide the UA the ability to manually invalidate individual temporary GRUUs. If a UA insists on not receiving any such inbound requests (including ones generated by network applications, such as those used for collecting digits), the UA can place a non-GRUU into the Contact header field. However, this is NOT RECOMMENDED. Usage of a GRUU coupled with automated call rejection features is far superior.

- As long as a temporary GRUU is used to populate the Contact header field, a correspondent will not be able to ascertain any information about the AOR or instance ID of the UA by inspection of the Contact header field. However, absent a network-provided privacy service, the IP address in the SDP can be used to determine information about the UA, such as its geographic location and ISP.

- In all cases, regardless of whether the UA uses a temporary or public GRUU in the Contact, regardless of whether it utilizes GRUU at all, and regardless of whether it invokes a network-provided privacy service, a correspondent will be able to determine the SIP

service provider of the UA.

11. IANA Considerations

This specification defines two new Contact header field parameters, one SIP URI parameter, and a SIP option tag.

11.1. Header Field Parameter

This specification defines two new header field parameters, as per the registry created by RFC 3968 [8]. The required information is as follows:

Header field in which the parameter can appear: Contact
Name of the Parameter: pub-gruu
Predefined Values: none
RFC Reference: RFC 5627

Header field in which the parameter can appear: Contact
Name of the Parameter: temp-gruu
Predefined Values: none
RFC Reference: RFC 5627

11.2. URI Parameter

This specification defines one new SIP URI parameter, as per the registry created by RFC 3969 [9].

Name of the Parameter: gr
Predefined Values: none
RFC Reference: RFC 5627

11.3. SIP Option Tag

This specification registers a new SIP option tag, as per the guidelines in Section 27.1 of RFC 3261 [1].

Name: gruu

Description: This option tag is used to identify the Globally Routable User Agent URI (GRUU) extension. When used in a Supported header, it indicates that a User Agent understands the extension. When used in a Require header field of a REGISTER request, it indicates that the registrar is not expected to process the registration unless it supports the GRUU extension.

12. Acknowledgments

The author would like to thank Eric Rescorla, Robert Sparks, Rohan Mahy, Paul Kyzivat, Alan Johnston, Ya-Ching Tan, Dale Worley, Jeroen van Bemmel, Vijay Gurbani, Andrew Allen, Alan Hawrylyshen, Francois Audet, Fredrik Thulin, Dean Willis, David Hancock, Keith Drage, and Cullen Jennings for their comments and contributions to this work. Eric Rescorla provided the text for the introduction and the GRUU construction algorithm in the appendix.

13. References

13.1. Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[3] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[6] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[7] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[8] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

[9] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, December 2004.

[10] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[11] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

[12] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[13] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[14] Jennings, C., Ed. and R. Mahy, Ed., "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

13.2. Informative References

[15] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[16] Rosenberg, J., Schulzrinne, H., and R. Mahy, "An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", RFC 4235, November 2005.

[17] Sparks, R., Hawrylyshen, A., Johnston, A., Rosenberg, J., and H. Schulzrinne, "Session Initiation Protocol (SIP) Torture Test Messages", RFC 4475, May 2006.

[18] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002.

[19] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.

[20] Burger, E. and M. Dolly, "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)", RFC 4730, November 2006.

[21] Mahy, R. and D. Petrie, "The Session Initiation Protocol (SIP) "Join" Header", RFC 3911, October 2004.

[22] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, September 2004.

[23] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

[24] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[25] Camarillo, G., "Compressing the Session Initiation Protocol (SIP)", RFC 3486, February 2003.

[26] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[27] Jennings, C., Audet, F., and J. Elwell, "Session Initiation Protocol (SIP) URIs for Applications such as Voicemail and Interactive Voice Response (IVR)", RFC 4458, April 2006.

[28] Kyzivat, P., "Registration Event Package Extension for Session Initiation Protocol (SIP) Globally Routable User Agent URIs (GRUUs)", RFC 5628, October 2009.

[29] Rosenberg, J., van Elburg, J., Holmberg, C., Audet, F., and S. Schubert, Ed., "Delivery of Request-URI Targets to User Agents", Work in Progress, June 2009.

Appendix A. Example GRUU Construction Algorithms

The mechanism for constructing a GRUU is not subject to specification. This appendix provides an example that can be used by a registrar to construct a public and a temporary GRUU. Of course, others are permitted, as long as they meet the constraints defined for a GRUU.

A.1. Public GRUU

The most basic approach for constructing a public GRUU is to take the AOR and place the actual value of the instance ID into the contents of the "gr" URI parameter.

A.2. Temporary GRUU

This specification requires a registrar to create a new temporary GRUU on each registration refresh. If a registration is very long lived, this can quickly result in hundreds or even thousands of temporary GRUUs being created and allocated to a UA. Consequently, it is important to have an algorithm for constructing temporary GRUUs that does not require additional storage that grows in size with the number of temporary GRUUs. The following algorithm meets this goal.

The registrar maintains a counter, I. This counter is 48 bits and is initialized to zero. The counter is persistently stored, using a backend database or other similar technique. When the registrar creates the first temporary GRUU for a particular AOR and instance ID, the registrar notes the current value of the counter, I_i, and increments the counter in the database. The registrar then maps I_i to the AOR and instance ID using the database, a persistent hashmap or similar technology. If the registration expires such that there are no longer any contacts with that particular instance ID bound to the GRUU, the registrar removes the mapping. Similarly, if the temporary GRUUs are invalidated due to a change in Call-ID, the registrar removes the current mapping from I_i to the AOR and instance ID, notes the current value of the counter I_j, and stores a mapping from I_j to the AOR and instance ID. Based on these rules, the hashmap will contain a single mapping for each AOR and instance ID for which there is a currently valid registration.

The usage of a counter in a 48-bit space with sequential assignment allows for a compact representation of the hashmap key, which is important for generating GRUUs of reasonable size. The counter starts at zero when the system is initialized. Persistent and reliable storage of the counter is required to avoid misrouting of a GRUU to the wrong AOR and instance ID. Similarly, persistent storage of the hashmap is required, even through proxy and registrar

restarts. If the hashmap is reset, all previous temporary GRUUs become invalidated. This might cause dialogs in progress to fail, or future requests towards a temporary GRUU to fail when they normally would not. The same hashmap needs to be accessible by all proxies and registrars that can field requests for a particular AOR and instance ID.

The registrar maintains a pair of local symmetric keys K_e and K_a. These are regenerated every time the counter is reset. When the counter rolls over or is reset, the registrar remembers the old values of K_e and K_a for a time. Like the hashmap itself, these keys need to be shared across all proxy and registrars that can service requests for a particular AOR and instance ID.

To generate a new temporary GRUU, the registrar generates a random 80-bit distinguisher value D. It then computes:

```
M = D || I_i
E = AES-ECB-Encrypt(K_e, M)
A = HMAC-SHA256-80(K_a, E)

Temp-Gruu-userpart = "tgruu." || base64(E) || base64(A)
```

where || denotes concatenation, and AES-ECB-Encrypt represents AES encryption in electronic codebook mode. M will be 128 bits long, producing a value of E that is 128 bits and A that is 80 bits. This produces a user part which has 42 characters.

When a proxy receives a request whose user part begins with "tgruu.", it extracts the remaining portion, and splits it into 22 characters (E') and the remaining 14 characters (A'). It then computes A and E by performing a base64 decode of A' and E' respectively. Next, it computes:

```
Ac = HMAC-SHA256-80(K_a, E)
```

If the counter has rolled over or reset, this computation is performed with the current and previous K_a. If the Ac value(s) that are computed do not match the value of A extracted from the GRUU, the GRUU is rejected as invalid. Next, the proxy computes:

```
M = AES-ECB-Decrypt(K_e, E)
```

If the counter has rolled over, this computation is done using the value of K_e that goes with the value of K_a, which produced a valid Ac in the previous HMAC validation. The leading 80 bits (the distinguisher D) are discarded, leaving an index I_i in the hashmap. This index is looked up. If it exists, the proxy now has the AOR and

instance ID corresponding to this temporary GRUU. If there is nothing in the hashmap for the key I_i, the GRUU is no longer valid and the request is rejected.

The usage of a 48-bit counter allows for the registrar to have as many as a billion AORs, with 10 instances per AOR, and cycle through 10,000 Call-ID changes for each instance through the duration of a single registration. These numbers reflect the average; the system works fine if a particular AOR has more than 10 instances or a particular instance cycles through more than 10,000 Call-IDs in its registration, as long as the average meets these constraints.

Appendix B. Network Design Considerations

The GRUU specification works properly based on logic implemented at the user agents and in the authoritative proxies on both sides of a call. Consequently, it is possible to construct network deployments in which GRUUs will not work properly.

One important assumption made by the GRUU mechanism is that, if a request passes through any proxies in the originating domain prior to visiting the terminating domain, one of those proxies will be the authoritative proxy for the User Agent Client (UAC). Administrators of SIP networks will need to make sure that this property is retained. There are several ways it can be accomplished:

1. If the user agents support the service-route mechanism [23], the registrar can implement it and return a service route that points to the authoritative proxy. This will cause requests originated by the user agent to pass through the authoritative proxy.

2. The user agents can be configured to never use an outbound proxy, and send requests directly to the domain of the terminating party. This configuration is not practical in many use cases, but it is a solution to this requirement.

3. The user agents can be configured with an outbound proxy in the same domain as the authoritative proxy, and this outbound proxy forwards requests to the authoritative proxy by default. This works very well in cases where the clients are not roaming; in such cases, the outbound proxy in a visited network may be discovered dynamically through DHCP [18].

4. In cases where the client discovers a local outbound proxy via a mechanism such as DHCP, and is not implementing the service route mechanism, the UA can be configured to automatically add an additional Route header field after the outbound proxy, which points to a proxy in the home network. This has the same net

effect of the service route mechanism, but is accomplished through static configuration.

Author's Address

Jonathan Rosenberg
Cisco Systems
Edison, NJ
US

EMail: jdrosen@cisco.com
URI: http://www.jdrosen.net