Internet Engineering Task Force (IETF)
Request for Comments: 8076
Category: Standards Track
ISSN: 2070-1721

A. Knauf
T. Schmidt, Ed.
HAW Hamburg
G. Hege
daviko GmbH
M. Waehlisch
link-lab & FU Berlin

March 2017

# A Usage for Shared Resources in RELOAD (ShaRe)

## Abstract

This document defines a REsource LOcation And Discovery (RELOAD) Usage for managing shared write access to RELOAD Resources. Shared Resources in RELOAD (ShaRe) form a basic primitive for enabling various coordination and notification schemes among distributed peers. Access in ShaRe is controlled by a hierarchical trust delegation scheme maintained within an access list. A new USER-CHAIN-ACL access policy allows authorized peers to write a Shared Resource without owning its corresponding certificate. This specification also adds mechanisms to store Resources with a variable name that is useful whenever peer-independent rendezvous processes are required.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8076.

## Copyright Notice

Table of Contents

## 1. Introduction

[RFC6940] defines the base protocol for REsource LOcation And Discovery (RELOAD), which allows for application-specific extensions by Usages. The present document defines such a RELOAD Usage for managing shared write access to RELOAD Resources and a mechanism to store Resources with variable names. The Usage for Shared Resources in RELOAD (ShaRe) enables overlay users to share their exclusive write access to specific Resource/Kind pairs with others. Shared Resources form a basic primitive for enabling various coordination and notification schemes among distributed peers. Write permission is controlled by an Access Control List (ACL) Kind that maintains a chain of Authorized Peers for a particular Shared Resource. A newly defined USER-CHAIN-ACL access control policy enables shared write access in RELOAD.

The Usage for Shared Resources in RELOAD is designed for jointly coordinated group applications among distributed peers (e.g., third-party registration, see [RFC7904], or distributed conferencing). Of particular interest are rendezvous processes, where a single identifier is linked to multiple, dynamic instances of a distributed cooperative service. Shared write access is based on a trust delegation mechanism that transfers the authorization to write a specific Kind data by storing logical Access Control Lists. An ACL contains the ID of the Kind to be shared and contains trust delegations from one authorized to another (previously unauthorized) user.

Shared write access augments the RELOAD security model, which is based on the restriction that peers are only allowed to write resources at a small set of well-defined locations (Resource-IDs) in the overlay. Using the standard access control rules in RELOAD, these locations are bound to the username or Node-ID in the peer's certificate. This document extends the base policies to enable a controlled write access for multiple users to a common Resource-ID.

Additionally, this specification defines an optional mechanism to store Resources with a variable Resource Name. It enables the storage of Resources whose name complies to a specific pattern. Definition of the pattern is arbitrary, but it must contain the username of the Resource creator.

## 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 [RFC2119].

This document uses the terminology and definitions from the RELOAD base [RFC6940] and [RFC7890], in particular the RELOAD Usage, Resource, and Kind. Additionally, the following terms are used:

Shared Resource: The term "Shared Resource" in this document defines a RELOAD Resource with its associated Kinds that can be written or overwritten by multiple RELOAD users following the specifications in this document.

Access Control List: The term "Access Control List" in this document defines a logical list of RELOAD users allowed to write a specific RELOAD Resource/Kind pair by following the specifications in this document. The list items are stored as Access Control List Kinds that map trust delegations from user A to user B, where A is allowed to write a Shared Resource and the Access Control List, while B is a user that obtains write access to specified Kinds from A.

Resource Owner: The term "Resource Owner" in this document defines a RELOAD peer that initially stored a Resource to be shared. The Resource Owner possesses the RELOAD certificate that grants write access to a specific Resource/Kind pair using the RELOAD certificate-based access control policies.

Authorized Peer: The term "Authorized Peer" in this document defines a RELOAD peer that was granted write access to a Shared Resource by permission of the Resource Owner or another Authorized Peer.

## 3. Shared Resources in RELOAD

A RELOAD user that owns a certificate for writing at a specific overlay location can maintain one or more RELOAD Kinds that are designated for a non-exclusive write access shared with other RELOAD users. The mechanism to share those Resource/Kind pairs with a group of users consists of two basic steps:

1. Storage of the Resource/Kind pairs to be shared.

2. Storage of an Access Control List (ACL) associated with those Kinds.

ACLs are created by the Resource Owner and contain ACL items, each delegating the permission of writing the shared Kind to a specific user called the "Authorized Peer". For each shared Kind data, its Resource owner stores a root item that initiates an Access Control List. Trust delegation to the Authorized Peer can include the right to further delegate the write permission, enabling a tree of trust delegations with the Resource Owner as trust anchor at its root.

The Resource/Kind pair to be shared can be any RELOAD Kind that complies to the following specifications:

Isolated Data Storage: To prevent concurrent writing from race conditions, each data item stored within a Shared Resource SHALL be exclusively maintained by the RELOAD user who created it. Hence, Usages that allow the storage of Shared Resources are REQUIRED to use either the array or dictionary data model and apply additional mechanisms for isolating data as described in Section 3.1.

Access Control Policy: To ensure write access to Shared Resource by Authorized Peers, each Usage MUST use the USER-CHAIN-ACL access policy as described in Section 6.6.

Resource Name Extension: To enable Shared Resources to be stored using a variable resource name, this document defines an optional ResourceNameExtension structure. It contains the Resource Name of the Kind data to be stored and allows any receiver of a shared data to validate whether the Resource Name hashes to the Resource-ID. The ResourceNameExtension is made optional by configuration. The ResourceNameExtension field is only present in the Kind data structure when configured in the corresponding kind-block of the overlay configuration document (for more details, see Section 5.3). If the configuration allows variable resource names, a Kind using the USER-CHAIN-ACL policy MUST use the ResourceNameExtension as the initial field within the Kind data structure definition. Otherwise, the Kind data structure does not contain the ResourceNameExtension structure.

## 3.1. Mechanisms for Isolating Stored Data

This section defines mechanisms to avoid race conditions while concurrently writing an array or dictionary of a Shared Resource.

If a dictionary is used in the Shared Resource, the dictionary key MUST be the Node-ID of the certificate that will be used to sign the stored data. Thus, data access is bound to the unique ID holder, and write concurrency does not occur.

If the data model of the Shared Resource is an array, each Authorized Peer that chooses to write data SHALL obtain its exclusive range of the array indices. The following algorithm will generate an array indexing scheme that avoids collisions:

1. Obtain the Node-ID of the certificate that will be used to sign the stored data.

2. Take the least significant 24 bits of that Node-ID to prefix the array index.

3. Append an 8-bit individual index value to those 24 bits of the Node-ID.

The resulting 32-bit long integer MUST be used as the index for storing an array entry in a Shared Resource. The 24 bits of the Node-ID serve as a collision-resistant identifier. The 8-bit individual index remains under the control of a single Peer and can be incremented individually for further array entries. In total, each Peer can generate 256 distinct entries for application-specific use.

The mechanism to create the array index inherits collision-resistance from the overlay hash function in use (e.g., SHA-1). It is designed to work reliably for small sizes of groups as applicable to resource sharing. In the rare event of a collision, the Storing Peer will refuse to (over-)write the requested array index and protect indexing integrity as defined in Section 6.1. A Peer could rejoin the overlay with a different Node-ID in such a case.

# 4. Access Control List Definition

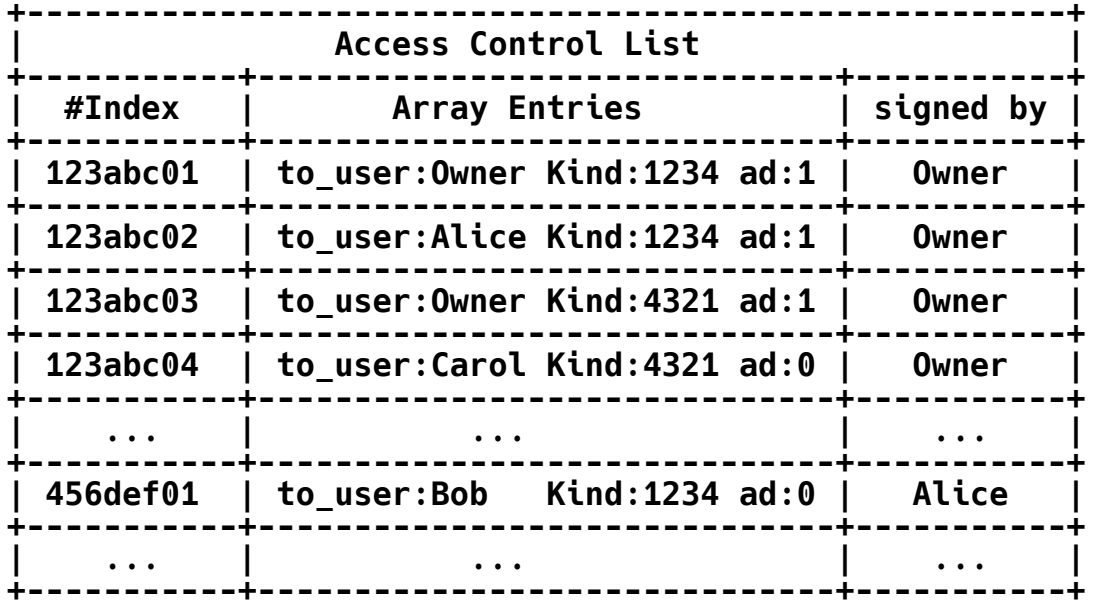
## 4.1. Overview

An Access Control List (ACL) is a (self-managed) Shared Resource that contains a list of AccessControlListItem structures as defined in Section 4.2. Each entry delegates write access for a specific Kind data to a single RELOAD user. An ACL enables the RELOAD user who is authorized to write a specific Resource-ID to delegate his exclusive write access to a specific Kind to further users of the same RELOAD overlay. Therefore, each Access Control List data structure carries the information about who obtains write access, the Kind-ID of the Resource to be shared, and whether delegation includes write access to the ACL itself. The latter condition grants the right to delegate write access further for the Authorized Peer. Access Control Lists are stored at the same overlay location as the Shared Resource and use the RELOAD array data model. They are initially created by the Resource Owner.

Figure 1 shows an example of an Access Control List. We omit the res_name_ext field to simplify illustration. The array entry at index 0x123abc01 displays the initial creation of an ACL for a Shared Resource of Kind-ID 1234 at the same Resource-ID. It represents the root item of the trust delegation tree for this shared RELOAD Kind. The root entry MUST contain the username of the Resource owner in the "to_user" field and can only be written by the owner of the public key certificate associated with this Resource-ID. The allow_delegation (ad) flag for a root ACL item is set to 1 by default. The array index is generated by using the mechanism for isolating stored data as described in Section 3.1. Hence, the most significant 24 bits of the array index (0x123abc) are the least significant 24 bits of the Node-ID of the Resource Owner.

The array item at index 0x123abc02 represents the first trust delegation to an Authorized Peer that is thus permitted to write to the Shared Resource of Kind-ID 1234. Additionally, the Authorized peer Alice is also granted write access to the ACL as indicated by the allow_delegation flag (ad) set to 1. This configuration authorizes Alice to store further trust delegations to the Shared Resource, i.e., add items to the ACL. On the contrary, index 0x456def01 illustrates trust delegation for Kind-ID 1234, in which the Authorized Peer Bob is not allowed to grant access to further peers (ad = 0). Each Authorized Peer signs its ACL items by using its own signer identity along with its own private key. This allows other peers to validate the origin of an ACL item and makes ownership transparent.

To manage Shared Resource access of multiple Kinds at a single location, the Resource Owner can create new ACL entries that refer to another Kind-ID as shown in array entry index 0x123abc03. Note that overwriting existing items in an Access Control List with a change in the Kind-ID revokes all trust delegations in the corresponding subtree (see Section 6.2). Authorized Peers are only enabled to overwrite existing ACL item they own. The Resource Owner is allowed to overwrite any existing ACL item, but should be aware of its consequences on the trust delegation chain.



| Access Control List | | |
|---|---|---|
| #Index | Array Entries | signed by |
| 123abc01 | to_user:Owner Kind:1234 ad:1 | Owner |
| 123abc02 | to_user:Alice Kind:1234 ad:1 | Owner |
| 123abc03 | to_user:Owner Kind:4321 ad:1 | Owner |
| 123abc04 | to_user:Carol Kind:4321 ad:0 | Owner |
| ... | ... | ... |
| 456def01 | to_user:Bob Kind:1234 ad:0 | Alice |
| ... | ... | ... |

**Figure 1: Simplified Example of an Access Control List, Including Entries for Two Different Kind-IDs and Varying Delegation (AD) Configurations**

Implementors of ShaRe should be aware that the trust delegation in an Access Control List need not be loop free. Self-contained circular trust delegation from A to B and B to A are syntactically possible, even though not very meaningful.

## 4.2. Data Structure

The Kind data structure for the Access Control List is defined as follows:

```
struct {
     /* res_name_ext is optional, see documentation */
     ResourceNameExtension  res_name_ext;
     opaque                 to_user<0..2^16-1>;
     KindId                 kind;
     Boolean                allow_delegation;
} AccessControlListItem;
```

The AccessControlListItem structure is composed of:

res_name_ext: This optional field contains the Resource Name of a ResourceNameExtension (see Section 5.2) to be used by a Shared Resource with a variable resource name. This name is used by the storing peer for validating, whether a variable resources name matches one of the predefined naming pattern from the configuration document for this Kind. The presence of this field is bound to a variable resource name element in the corresponding kind-block of the configuration document whose "enable" attribute is set to true (see Section 5.3). Otherwise, if the "enable" attribute is false, the res_name_ext field SHALL NOT be present in the Kind data structure.

to_user: This field contains the username of the RELOAD peer that obtains write permission to the Shared Resource.

kind: This field contains the Kind-ID of the Shared Resource.

allow_delegation: If true, this Boolean flag indicates that the Authorized Peer in the 'to_user' field is allowed to add additional entries to the ACL for the specified Kind-ID.

# 5. Extension for Variable Resource Names

## 5.1. Overview

In certain use cases, such as conferencing, it is desirable to increase the flexibility of a peer in using Resource Names beyond those defined by the username or Node-ID fields in its certificate. For this purpose, this document presents the concept for variable Resources Names that enables providers of RELOAD instances to define relaxed naming schemes for overlay Resources.

Each RELOAD node uses a certificate to identify itself using its username (or Node-ID) while storing data under a specific Resource-ID (see Section 7.3 in [RFC6940]). The specifications in this document scheme adhere to this paradigm, but enable a RELOAD peer to store values of Resource Names that are derived from the username in its certificate. This is done by using a Resource Name with a variable substring that still matches the username in the certificate using a pattern defined in the overlay configuration document. Thus, despite being variable, an allowable Resource Name remains tied to the Owner's certificate. A sample pattern might be formed as follows:

Example Pattern:
.*-conf-$USER@$DOMAIN

When defining the pattern, care must be taken to avoid conflicts arising from two usernames of which one is a substring of the other. In such cases, the holder of the shorter name could threaten to block the resources of the longer-named peer by choosing the variable part of a Resource Name to contain the entire longer username. For example, a "*$USER" pattern would allow user EVE to define a resource with name "STEVE" and to block the resource name for user STEVE through this. This problem can easily be mitigated by delimiting the variable part of the pattern from the username part by some fixed string, that by convention is not part of a username (e.g., the "-conf-" in the above Example).

## 5.2. Data Structure

This section defines the optional ResourceNameExtension structure for every Kind that uses the USER-CHAIN-ACL access control policy.

```
enum { pattern(1), (255)} ResourceNameType;

struct {
  ResourceNameType type;
  uint16           length;

  select(type) {
      case pattern:
        opaque     resource_name<0..2^16-1>;

      /* Types can be extended */
  };
} ResourceNameExtension;
```

The content of the ResourceNameExtension consists of:

length: This field contains the length of the remaining data structure. It is only used to allow for further extensions to this data structure.

The content of the rest of the data structure depends of the ResourceNameType. Currently, the only defined type is "pattern".

If the type is "pattern", then the following data structure contains an opaque <0..2^16-1> field containing the Resource Name of the Kind being stored. The type "pattern" further indicates that the Resource Name MUST match to one of the variable resource name patterns defined for this Kind in the configuration document.

The ResourceNameType enum and the ResourceNameExtension structure can be extended by further Usages to define other naming schemes.

### 5.3. Overlay Configuration Document Extension

This section extends the overlay configuration document by defining new elements for patterns relating resource names to usernames. It is noteworthy that additional constraints on the syntax and semantic of names can apply according to specific Usages. For example, Address of Record (AOR) syntax restrictions apply when using P2PSIP [RFC7904], while a more general naming is feasible in plain RELOAD.

The <variable-resource-names> element serves as a container for one or multiple <pattern> sub-elements. It is an additional parameter within the kind-block and has a boolean "enable" attribute that indicates, if true, that the overlay provider allows variable resource names for this Kind. The default value of the "enable" attribute is "false". In the absence of a <variable-resource-names> element for a Kind using the USER-CHAIN-ACL access policy (see Section 6.6), implementors MUST assume this default value.

A <pattern> element MUST be present if the "enabled" attribute of its parent element is set to true. Each <pattern> element defines a pattern for constructing extended resource names for a single Kind. It is of type xsd:string and interpreted as a regular expression according to "POSIX Extended Regular Expression" (see the specifications in [IEEE-Posix]). In this regular expression, $USER and $DOMAIN are used as variables for the corresponding parts of the string in the certificate username field (with $USER preceding and $DOMAIN succeeding the '@'). Both variables MUST be present in any given pattern definition. Furthermore, variable parts in <pattern> elements defined in the overlay configuration document MUST remain syntactically separated from the username part (e.g., by a dedicated delimiter) to prevent collisions with other names of other users. If no pattern is defined for a Kind, if the "enable" attribute is false, or if the regular expression does not meet the requirements specified in this section, the allowable Resource Names are restricted to the username of the signer for Shared Resource.

The RELAX NG Grammar for the Variable Resource Names Extension reads:

```
# VARIABLE RESOURCE URN SUB-NAMESPACE

namespace share = "urn:ietf:params:xml:ns:p2p:config-base:share"

# VARIABLE RESOURCE NAMES ELEMENT

kind-parameter &= element share:variable-resource-names {

    attribute enable { xsd:boolean },

    # PATTERN ELEMENT

    element share:pattern { xsd:string }*
}?
```

Whitespace and case processing follows the rules of [OASIS.relax_ng] and XML Schema Datatypes [W3C.REC-xmlschema-2-20041028].

## 6. Access Control to Shared Resources

### 6.1. Granting Write Access

Write access to a Kind that is intended to be shared with other RELOAD users can be initiated solely by the Resource Owner. A Resource Owner can share RELOAD Kinds by using the following procedure:

- The Resource Owner stores an ACL root item at the Resource-ID of the Shared Resource. The root item contains the ResourceNameExtension field (see Section 5.2), the username of the Resource Owner and Kind-ID of the Shared Resource. The allow_delegation flag is set to 1. The index of array data structure MUST be generated as described in Section 3.1.

- Further ACL items for this Kind-ID stored by the Resource Owner MAY delegate write access to Authorized Peers. These ACL items contain the same resource name extension field, the username of the Authorized Peer, and the Kind-ID of the Shared Resource. Optionally, the Resource Owner sets the "ad" to 1 (the default equals 0) to enable the Authorized Peer to further delegate write access. For each succeeding ACL item, the Resource Owner increments its individual index value by one (see Section 3.1) so that items can be stored in the numerical order of the array index starting with the index of the root item.

An Authorized Peer with delegation allowance ("ad"=1) can extend the access to an existing Shared Resource as follows:

- An Authorized Peer can store additional ACL items at the Resource-ID of the Shared Resource. These ACL items contain the resource name extension field, the username of the newly Authorized Peer, and the Kind-ID of the Shared Resource. Optionally, the "ad" flag is set to 1 for allowing the newly Authorized Peer to further delegate write access. The array index MUST be generated as described in Section 3.1. Each succeeding ACL item can be stored in the numerical order of the array index.

A store request by an Authorized Peer that attempts to overwrite any ACL item signed by another Peer is unauthorized and causes an Error_Forbidden response from the Storing Peer. Such access conflicts could be caused by an array index collision. However, the probability of a collision of two or more identical array indices will be negligibly low using the mechanism for isolating stored data (see Section 3.1).

## 6.2. Revoking Write Access

Write permissions are revoked by storing a nonexistent value (see [RFC6940], Section 7.2.1) at the corresponding item of the Access Control List. Revoking a permission automatically invalidates all delegations performed by that user including all subsequent delegations. This allows the invalidation of entire subtrees of the delegations tree with only a single operation. Overwriting the root item with a nonexistent value of an Access List invalidates the entire delegations tree.

An existing ACL item MUST only be overwritten by the user who initially stored the corresponding entry, or by the Resource Owner that is allowed to overwrite all ACL items for revoking write access.

To protect the privacy of the users, the Resource Owner SHOULD overwrite all subtrees that have been invalidated.

## 6.3. Validating Write Access through an ACL

Access Control Lists are used to transparently validate authorization of peers for writing a data value at a Shared Resource. Thereby, it is assumed that the validating peer is in possession of the complete and most recent ACL for a specific Resource/Kind pair. The corresponding procedure consists of recursively traversing the trust delegation tree with strings compared as binary objects. It proceeds as follows:

1. Obtain the username of the certificate used for signing the data stored at the Shared Resource. This is the user who requested the write operation.

2. Validate that an item of the corresponding ACL (i.e., for this Resource/Kind pair) contains a "to_user" field whose value equals the username obtained in step 1. If the Shared Resource under examination is an Access Control List Kind, further validate if the "ad" flag is set to 1.

3. Select the username of the certificate that was used to sign the ACL item obtained in the previous step.

4. Validate that an item of the corresponding ACL contains a "to_user" field whose value equals the username obtained in step 3. Additionally, validate that the "ad" flag is set to 1.

5. Repeat steps 3 and 4 until the "to_user" value is equal to the username of the signer of the ACL in the selected item. This final ACL item is expected to be the root item of this ACL, which MUST be further validated by verifying that the root item was signed by the owner of the ACL Resource.

The trust delegation chain is valid if and only if all verification steps succeed. In this case, the creator of the data value of the Shared Resource is an Authorized Peer.

Note that the ACL validation procedure can be omitted whenever the creator of data at a Shared Resource is the Resource Owner itself. The latter can be verified by its public key certificate as defined in Section 6.6.

## 6.4. Operations of Storing Peers

Storing peers, at which Shared Resource and ACL are physically stored, are responsible for controlling storage attempts to a Shared Resource and its corresponding Access Control List. To assert the USER-CHAIN-ACL access policy (see Section 6.6), a storing peer MUST perform the access validation procedure described in Section 6.3 on any incoming store request using the most recent Access Control List for every Kind that uses the USER-CHAIN-ACL policy. It SHALL further ensure that only the Resource Owner stores new ACL root items for Shared Resources.

### 6.5. Operations of Accessing Peers

Accessing peers, i.e., peers that fetch a Shared Resource, can validate that the originator of a Shared Resource was authorized to store data at this Resource-ID by processing the corresponding ACL. To enable an accessing peer to perform the access validation procedure described in Section 6.3, it first needs to obtain the most recent Access Control List in the following way:

1. Send a Stat request to the Resource-ID of the Shared Resource to obtain all array indexes of stored ACL Kinds (as per [RFC6940], Section 7.4.3.).

2. Fetch all indexes of existing ACL items at this Resource-ID by using the array ranges retrieved in the Stat request answer.

Peers can cache previously fetched Access Control Lists up to the maximum lifetime of an individual item. Since stored values could have been modified or invalidated prior to their expiration, an accessing peer SHOULD use a Stat request to check for updates prior to using the data cache.

### 6.6. USER-CHAIN-ACL Access Policy

This document specifies an additional access control policy to the RELOAD base document [RFC6940]. The USER-CHAIN-ACL policy allows Authorized Peers to write a Shared Resource, even though they do not own the corresponding certificate. Additionally, the USER-CHAIN-ACL allows the storage of Kinds with a variable resource name that are following one of the specified naming patterns. Hence, on an inbound store request on a Kind that uses the USER-CHAIN-ACL access policy, the following rules MUST be applied:

In the USER-CHAIN-ACL policy, a given value MUST NOT be written or overwritten, if neither one of USER-MATCH or USER-NODE-MATCH (mandatory if the data model is dictionary) access policies of the base document [RFC6940] applies.

Additionally, the store request MUST be denied if the signer's certificate does not contain a username that matches to the user and domain portion in one of the variable resource name patterns (cf. Section 5) specified in the configuration document or if the hashed Resource Name does not match the Resource-ID. The Resource Name of the Kind to be stored MUST be taken from the mandatory ResourceNameExtension field in the corresponding Kind data structure.

If the access rights cannot be verified according to the ACL validation procedure described in Section 6.3, the store request MUST also be denied.

Otherwise, the store request can be processed further.

## 7. ACCESS-CONTROL-LIST Kind Definition

This section defines the ACCESS-CONTROL-LIST Kind previously described in this document.

Name: ACCESS-CONTROL-LIST

Kind IDs: The Resource Name for ACCESS-CONTROL-LIST Kind-ID is the Resource Name of the Kind that will be shared by using the ACCESS-CONTROL-LIST Kind.

Data Model: The data model for the ACCESS-CONTROL-LIST Kind-ID is array. The array indexes are formed by using the mechanism for isolated stored data as described in Section 3.1.

Access Control: USER-CHAIN-ACL (see Section 6.6).

## 8. Security Considerations

In this section, we discuss security issues that are relevant to the usage of Shared Resources in RELOAD [RFC6940].

### 8.1. Resource Exhaustion

Joining a RELOAD overlay inherently poses a certain resource load on a peer, because it has to store and forward data for other peers. In common RELOAD semantics, each Resource-ID and thus position in the overlay, may only be written by a limited set of peers -- often even only a single peer, which limits this burden. In the case of Shared Resources, a single resource may be written by multiple peers who may even write an arbitrary number of entries (e.g., delegations in the ACL). This leads to an enhanced use of resources at individual overlay nodes. The problem of resource exhaustion can easily be mitigated for Usages based on the ShaRe-Usage by imposing restrictions on size, i.e., <max-size> element for a certain Kind in the configuration document.

### 8.2. Malicious or Misbehaving Storing Peer

The RELOAD overlay is designed to operate despite the presence of a small set of misbehaving peers. This is not different for Shared Resources since a small set of malicious peers does not disrupt the functionality of the overlay in general, but may have implications for the peers needing to store or access information at the specific locations in the ID space controlled by a malicious peer. A storing peer could withhold stored data, which results in a denial of service to the group using the specific resource. But it could not return forged data, since the validity of any stored data can be independently verified using the attached signatures.

### 8.3. Trust Delegation to a Malicious or Misbehaving Peer

A Resource Owner that erroneously delegated write access to a Shared Resource for a misbehaving peer enables this malicious member of the overlay to interfere with the corresponding group application in several unwanted ways. Examples of destructive interferences range from exhausting shared storage to dedicated application-specific misuse. Additionally, a bogus peer that was granted delegation rights may authorize further malicious collaborators to writing the Shared Resource.

It is the obligation of the Resource Owner to bind trust delegation to apparent trustworthiness. Additional measures to monitor proper behavior may be applied. In any case, the Resource Owner will be able to revoke the trust delegation of an entire tree in a single overwrite operation. It further holds the right to overwrite any malicious contributions to the shared resource under misuse.

### 8.4. Privacy Issues

All data stored in the Shared Resource is readable by any node in the overlay; thus, applications requiring privacy need to encrypt the data. The ACL needs to be stored unencrypted; thus, the list members of a group using a Shared Resource will always be publicly visible.

# 9. IANA Considerations

## 9.1. Access Control Policy

IANA has registered the following entry in the "RELOAD Access Control Policies" registry (cf. [RFC6940]) to represent the USER-CHAIN-ACL Access Control Policy, as described in Section 6.6.

| Access Policy | RFC |
|---|---|
| USER-CHAIN-ACL | RFC 8076 |

## 9.2. Data Kind-ID

IANA has registered the following code point in the "RELOAD Data Kind-ID" registry (cf. [RFC6940]) to represent the ShaRe ACCESS-CONTROL-LIST kind, as described in Section 7.

| Kind | Kind-ID | RFC |
|---|---|---|
| ACCESS-CONTROL-LIST | 0x4 | RFC 8076 |

## 9.3. XML Namespace Registration

This document registers the following URI for the config XML namespace in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:p2p:config-base:share

Registrant Contact: The IESG

XML: N/A, the requested URI is an XML namespace

## 10. References

### 10.1. Normative References

[IEEE-Posix] "IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities (Vol. 1)", IEEE Std 1003.2-1992, ISBN 1-55937-255-9, DOI 10.1109/IEEESTD.1993.6880751, January 1993, <http://ieeexplore.ieee.org/document/6880751/>.

[OASIS.relax_ng] Clark, J. and M. Murata, "RELAX NG Specification", December 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC6940] Jennings, C., Lowekamp, B., Ed., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", RFC 6940, DOI 10.17487/RFC6940, January 2014, <http://www.rfc-editor.org/info/rfc6940>.

[W3C.REC-xmlschema-2-20041028] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

### 10.2. Informative References

[RFC7890] Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, "Concepts and Terminology for Peer-to-Peer SIP (P2PSIP)", RFC 7890, DOI 10.17487/RFC7890, June 2016, <http://www.rfc-editor.org/info/rfc7890>.

[RFC7904] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., Schulzrinne, H., and T. Schmidt, Ed., "A SIP Usage for REsource LOcation And Discovery (RELOAD)", RFC 7904, DOI 10.17487/RFC7904, October 2016, <http://www.rfc-editor.org/info/rfc7904>.

## Acknowledgments

This work was stimulated by fruitful discussions in the P2PSIP working group and the SAM research group. We would like to thank all active members for their constructive thoughts and feedback. In particular, the authors would like to thank (in alphabetical order) Emmanuel Baccelli, Ben Campbell, Alissa Cooper, Lothar Grimm, Russ Housley, Cullen Jennings, Matt Miller, Peter Musgrave, Joerg Ott, Marc Petit-Huguenin, Peter Pogrzeba, and Jan Seedorf. This work was partly funded by the German Federal Ministry of Education and Research, projects HAMcast, Mindstone, and SAFEST.

## Authors' Addresses

Alexander Knauf
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: alexanderknauf@gmail.com

Thomas C. Schmidt
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Email: t.schmidt@haw-hamburg.de
URI: http://inet.haw-hamburg.de/members/schmidt

Gabriel Hege
daviko GmbH
Schillerstr. 107
Berlin D-10625
Germany

Phone: +493043004344
Email: hege@daviko.com

Matthias Waehlisch
link-lab & FU Berlin
Hoenower Str. 35
Berlin D-10318
Germany

Email: mw@link-lab.net
URI: http://www.inf.fu-berlin.de/~waehl