AllJoyn™ Security 2.0 Feature

High-Level Design Document

Rev 1 Update 6 Draft 12

March 27, 2015

Contents

1 Introduction 5

1.1 Purpose and scope 5

1.2 Revision history 5

1.3 Acronyms and terms 5

2 System Design 8

2.1 Overview 8

2.2 Premises 9

2.3 Typical operations 11

2.3.1 Assumptions 11

2.3.2 Sample Certificates and Policy Entries 11

2.3.3 Define a security group 12

2.3.4 Required Key Exchanges 13

2.3.5 Certificate exchange during session establishment 13

2.3.6 Claim a factory-reset application 15

2.3.7 Example of building a policy 18

2.3.8 Install a policy 19

2.3.9 Install a manifest 19

2.3.10 Add an application to a security group 20

2.3.11 Add a user to a security group 21

2.3.12 Security Manager 22

2.3.13 Application Manifest 31

2.4 Access validation 34

2.4.1 Validating policy on a producer 34

2.4.2 Validating policy on a consumer 35

2.4.3 Validating policy on a consumer that requires producers membership in a security group 36

2.4.4 Anonymous session 37

2.4.5 Validating an admin 39

2.4.6 Emitting a session-based signal 39

2.5 Authorization data format 40

2.5.1 The format is binary and exchanged between peers using AllJoyn marshalling 40

2.5.2 Format Structure 41

2.6 Certificates 46

2.6.1 Main Certificate Structure 46

2.6.2 Identity certificate 46

2.6.3 Membership certificate 47

2.7 Sample use cases 48

2.7.1 Users and devices 48

2.7.2 Users set up by Dad 49

2.7.3 Living room set up by Dad 50

2.7.4 Son's bedroom set up by son 51

2.7.5 Master bedroom set up by Dad 52

2.7.6 Son can control different TVs in the house 53

2.7.7 Living room tablet controls TVs in the house 54

3 Enhancements to Existing Framework 55

3.1 Crypto Agility Exchange 55

3.2 Permission NotifyConfig Announcement 56

4 Features In Future Releases 57

4.1.1 Certificate revocation (not fully designed) 57

4.1.2 Distribution of policy updates and membership certificates (not fully designed) 57

4.1.3 Policy Templates 58

5 Future Considerations 58

5.1 Broadcast signals and multipoint sessions 58

Figures

Figure 2‑1. Security system diagram 9

Figure 2‑2: Sample Certificates and ACL entries 12

Figure 2‑3. Exchange manifest and membership certificates 14

Figure 2‑4: Claim a factory-reset application without using out-of-band registration data 16

Figure 2‑5. Claiming a factory-reset application using out-of-band registration data 18

Figure 2‑6. Install a policy 19

Figure 2‑7: Install manifest 20

Figure 2‑8. Add an application to a security group 21

Figure 2‑9. Add a user to a security group 22

Figure 2‑10: Building Policy using manifest 33

Figure 2‑11. Validating policy on a producer 35

Figure 2‑12. Validating policy for a consumer 36

Figure 2‑13: Consumer policy requires producer belong to a security group 37

Figure 2‑14. Anonymous access 38

Figure 2‑15. Validating an admin 39

Figure 2‑16. Validating a session-based signal 40

Figure 2‑17: Authorization Data Format Structure 41

Figure 2‑18. Use case - users set up by Dad 49

Figure 2‑19. Use case - living room set up by Dad 50

Figure 2‑20. Use case - son's bedroom set up by son 51

Figure 2‑21. Use case - master bedroom set up by Dad 52

Figure 2‑22. Use case – Son can control different TVs in the house 53

Figure 2‑23. Use case - Living room tablet controls TVs 54

Figure 4‑1. Distribution of policy update and certificates 58

Tables

Table 2‑1. Security 2.0 premises 9

Table 2-2: Action Mask Matrix 43

# Introduction

## Purpose and scope

This document captures the system level design for the enhancements to the AllJoyn™ framework to support the Security 2.0 feature requirements. Related interfaces and API design is captured at a functional level. Actual definition for interfaces and APIs is outside the scope of this document. Features and functions are subject to change without notice.

## Revision history

|  |  |  |
| --- | --- | --- |
| Revision | Date | Change Log |
| Rev 1 Update 0 | August 8, 2014 | Update with new format and comments |
| Rev 1 Update 1 | August 27, 2014 | Update with comments from the collaboration meeting |
| Rev 1 Update 2 | September 8, 2014 | Update with comments and agreement from the technical conference call on September 3, 2014. |
| Rev 1 Update 3 | October 30, 2014 | Update the authorization data section based on agreement from the technical conference call on October 14, 2014. |
| Rev 1 Update 4 | December 23, 2014 | Update the Certificate section and changes listed in JIRA tickets ASACORE-1170, 1256, 1259, 1260. |
| Rev 1 Update 5 | January 15, 2015 | Update the rule enforcing table after the conference call on Janurary 13, 2015 by the Security2.0 working group. |
| Rev 1 Update 6 | March 23, 2015 | Update the authorization data after the conference call on Janurary 20, 2015 by the Security2.0 working group. Updated the permission matrix to reflect the concept of Provide permission.  Updated based on review comments by the Security 2.0 working group on March 6, 2015.  Add the updated information on Security Group equivalence and manifest from the Wiki  Updated based on review comments by the Security 2.0 working group on March 13, 2015 and on March 19, 2015.  Updated based on open issue discussion on March 23, 2015. |

## Acronyms and terms

| Acronym/term | Description |
| --- | --- |
| About data | Data from the About feature. For more information, refer to the [About Feature Interface Spec](https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-about-feature-10-interface-specification). |
| ACL | Access Control List |
| AES CCM | The Advanced Encryption Standard 128-bit block cypher using Counter with CBC-MAC mode. Refer to [RFC 3610](http://tools.ietf.org/html/rfc3610) for more information. |
| AllJoyn framework | Open source peer-to-peer framework that allows for abstraction of low-level network concepts and APIs. |
| Authentication GUID | The Authentication GUID is a GUID assigned to a keystore for authentication purposes. This GUID is persisted in the keystore and provides a long-term identity for the application. Typically, this GUID is associated with a single application. In the scenario where a group of related applications share a given key store, they also share the same authentication GUID.  This GUID is used as a mapping key for storing and accessing authentication and encryption keys. All key materials associated with another peer is stored in the key store with the peer’s authentication GUID as the mapping key. |
| Certificate Authority (CA) | Entity that issues a digital certificate |
| Consumer | An AllJoyn application consuming services on the AllJoyn network. |
| Device | A physical device that may contain one or more AllJoyn applications. In this document, whenever the term “device” is used, it indicates the system application of the given physical device. |
| DSA | Digital Signature Algorithm |
| ECC | Elliptic Curve Cryptography |
| ECDHE | Elliptic Curve Diffie-Hellman Ephemeral key exchange |
| ECDHE\_ECDSA | ECDHE key agreement with asymmetric DSA based authentication. |
| ECDHE\_NULL | ECDHE key agreement only. No authentication. |
| ECDHE\_PSK | ECDHE key agreement with symmetric key/pin/password based authentication. |
| Factory-reset application | An application is restored to the original configuration. |
| Friend | A user who has a trusted relationship with the owner |
| Grantee | The application or user who is the subject of a certificate. |
| GUID | Globally Unique Identifier. A 128 bit identifier generated randomly in a way that the probability of collision is negligible. |
| Holder | The application or user possessing a certificate. |
| Keystore | A repository of security keys and certificates. An application instance can have at least one keystore. A keystore is associated with a bus attachment. If an application uses multiple bus attachments, it can have more than one keystores. The practice of using multiple bus attachments is discouraged because of complex side effect.  Multiple applications running as the same user can choose to share the same keystore, but if they do, they are considered as if they were the same application since they are treated as the same security principal. |
| OOB | Out Of Band |
| Peer | A remote application participating in the AllJoyn messaging. |
| Permission Management module | The AllJoyn Core module that handles all the permission authorization. |
| Producer | An AllJoyn application providing services on the AllJoyn network. |
| Security Appliance | A security appliance is a type of Security Manager that is always on. |
| Security Group | A logical grouping of devices, applications, and users. It is identified by a group ID which is a GUID and the security group authority public key. An application can be installed with a policy to expose services to members of the security group. An application or user holding a membership certificate is in fact a member of the security group. Any member of the security group can access the services exposed to the group by the applications with ACLs defined for that group. |
| Security Group Authority | A security group authority is the user or application that defines the security group and grant membership certificates to other. The security group authority is the certificate authority for that group. |
| Security Manager | An application used to manage cryptographic keys, and generate and distribute certificates. |
| SHA-256 | Secure Hash Algorithm SHA-2 with digest size of 256 bits or 32 bytes. |
| User | The person or business entity interacting with AllJoyn applications. |

# System Design

## Overview

The goal of the Security 2.0 feature is to allow an application to validate access to interfaces or objects based on policies installed by the owner. This feature is part of the AllJoyn Core library. It is not an option for the application to enforce permission. It is up to the user to dictate how the application performs based on the access control lists (ACLs) defined for the application. The AllJoyn Core Permission Management component does all the enforcement including the concept of mutual or one-way authorization before any message action can be taken.

The Security Manager is a service that helps the user with key management and permission rules building. Using application manifest defined by an application developer, the Security Manager builds the access control lists to let the end-user authorize which interactions the application can do. An application developer does not have to build a security manager. The permission can be installed by another application or another security manager.

In addition to the encrypted messaging (using AES CCM) between the peers, the Security 2.0 Permission Management module manages a database of access credentials and the Access Control Lists (ACLs).

Figure 2‑1 shows the system architecture of the Security 2.0 feature.



Figure 2‑1. Security system diagram

## Premises

Table 2‑1 lists the premises for the Security 2.0 features.

Table ‑. Security 2.0 premises

| Topic | Definition | Premises |
| --- | --- | --- |
| Identity | The application security principal | Each peer is identified by an authentication GUID and a cryptographic public key |
| Admin | An admin (or administrator) is a security principal with administrator privilege for the application | * An admin has full access to any object and interface in the application * An admin becomes a certificate authority * An admin can be a public key or a security group |
| Claim | Incorporate a factory-reset application with the Permission Management | * A factory-reset application has no list of certificate authorities for AllJoyn security. * A factory-reset application has no admin for AllJoyn security. * Anyone can claim as an admin for a factory-reset application. |
| Policy | A policy is a list of rules governing the behavior of an application  A policy template is a list of rules defined by the application developer to guide the user for policy building. | * An admin can install, update, or remove a policy. * A newer policy can be installed by any authorized peer. Developers can define policy templates to help the admin with policy building. * Security group specific policy specifies the permissions granted to members of the group. The security group authority becomes a certificate authority for that particular group. * A policy may exist at the producer or consumer side. Policy enforcement applies wherever it resides. * A policy is considered private. It is not exchanged with any peer. * A keystore has at most one policy. A complex application with multiple bus attachments can use a shared keystore in one bus attachment and an app-specific keystore for another bus attachment. In such case, the complex application has in fact more than one policies. * An admin can query the existing policy installed in the keystore. |
| Membership certificate | A membership certificate is the proof of a security group membership | * Membership certificates are exchanged between peers. The authorization data signed by this certificate are used for mutual authorization purposes. * An application trusts a membership certificate if the issuer or any subject in the issuer’s certificate chain is the owner or the security group authority. * A membership certificate holder can generate additional membership certificates for the given security group with the same or more restrictive permissions if the delegate flag is enabled. This type of membership certificate will not allow further delegation. * A membership certificate must have a security group ID. * An application can accept the installation of any number of membership certificates into its keystore. |
| Identity certificate | Certificate that signs the identity information. | * The Certificate has an identity alias stored in the X.509 SubjectAltName extension field. * An application trusts identity certificate issued by the owner, any of the application’s certificate authorities, or any of the security group authorities listed in the application’s policy. |
| Manifest data | The permission rules accompanying the identity certificate | * Manifest data are not present in the identity certificate. They are accompanied with the identity certificate. * The manifest data digest is present in the identity certificate. * The manifest data syntax is the same as the policy syntax. While the policy stays local the manifest data is presented to the peer along with the membership certificate. |
| Identity certificate | Certificate that signs the identity information. | * Certificate with a digest of the actual identity data. The identity data can be delivered out of band. * The Certificate has an identity alias stored in the X.509 SubjectAltName extension field. * An application trusts identity certificate issued by the owner or any of the application’s certificate authorities and security group equivalence authorities. |
| Security Manager | An application used to manage cryptographic keys, and generate certificates. | * Security Manager can push policy and certificates to application |

## Typical operations

The following subsections describe the typical operations performed by a user.

### Assumptions

In all the flows listed in this section, the Security Manager is assumed to be claimed by another Security Manager or to be self-claimed. The certificates may have been issued from sources in the cloud. As the result, the Security Manager is shown with one certificate authority and an identify certificate.

### Sample Certificates and Policy Entries

The following is a high level presentation of certificates and policy entries used in the flows in this section.



Figure 2‑2: Sample Certificates and ACL entries

### Define a security group

Any user can define a security group (logical grouping of applications and users) using a Security Manager. When the user specifies a security group name (for display purpose), the Security Manager creates the security group ID (a GUID value).

### Required Key Exchanges

The framework requires either ECHDE\_NULL or ECDHE\_PSK key exchange for the claim process. Once the application is claimed, only the ECDHE\_ECDSA key exchange is allowed unless the policy allows for anonymous user; in such case, ECDHE\_NULL is acceptable.

### Certificate exchange during session establishment

During the AllJoyn ECHDE\_ECDSA key exchange and session establishment, the peers exchange identity certificate, manifest data, and all membership certificates. Since all the membership certificates are exchanged, there is a potential information disclosure vulnerability. It is desired to have a more intelligent selection algorithm to provide membership certificates on demand and need-to-know basis. This algorithm needs to take into account of the latency of the certificate exchange during the method call invocation.

During the ECDHE\_ECDSA key exchange phase, the Identity certificate is exchanged. The application trusts the peer if the issuer of the peer’s identity certificate is the owner or any of the application certificate authorities.

After the session key is generated, the peers exchange all the membership certificates. Each membership certificate’s trust is checked against the owner’s public key or the public key of any of the security group authorities.



Figure 2‑3. Exchange manifest and membership certificates

### Claim a factory-reset application

Using a Security Manager any user can claim any factory-reset application. The factory-reset application is assumed to be already onboarded to the network. Claiming is a first-come, first-claim action. That user becomes the admin. The user also installs an admin security group. The procedure to make the application to become claimable again is manufacturer-specific. There will be an API call that allows the application to make itself claimable again.

#### Claim factory-reset application without out-of-band registration data

Figure 2‑4: Claim a factory-reset application without using out-of-band registration data

The identity certificate will be used for authentication in the ECDHE\_ECDSA key exchange.

#### Claim factory-reset application using out-of-band registration data

An application manufacturer can provision a key to support the claiming process. The ECDHE\_PSK key exchange is used in this scenario. The key is provided to the user out of band. An example is a QR code or a token delivered via email or text messaging. The user is prompted for the key when establishing a connection with the factory-reset application.



Figure 2‑5. Claiming a factory-reset application using out-of-band registration data

### Example of building a policy

A user uses a Security Manager application to build a policy. The Security Manager application queries the About data and manifest data from the application. The Security Manager application can do further introspection of the application for the detailed information of securable interfaces and secured objects, and prompt the user to select the permissions to include in the policy.

A policy may contain a number of ACLs. Please refer to section 2.5 (*Authorization data format)* for more information.

### Install a policy

An admin can install a policy for the application.



Figure 2‑6. Install a policy

### Install a manifest

An admin can install a manifest for the application.



Figure 2‑7: Install manifest

### Add an application to a security group

An admin issues a membership certificate with the given security group ID and installs it in the application’s keystore. This act adds the application to the security group.



Figure 2‑8. Add an application to a security group

### Add a user to a security group

The security group authority uses the Security Manager to generate the membership certificate for the user for the given security group ID.



Figure 2‑9. Add a user to a security group

### Security Manager

#### Introduction

The AllJoyn security 2.0 ecosystem consists of many applications and devices. Those applications and devices are deployed in various setups and for them it is impossible to know up front what other peers they will see around them let alone know how they should interact with them. Which peers can be trusted, which rights do those peers have… So after being deployed, applications and devices have to be configured. The people in charge of configuring the system, the administrators need a tool for this. Such a tool is called a security manager.

Depending on your setup, you need a different tool. A large enterprise has different requirements than a home does. Not all administrators have a strong technical background. A tool for home users should have a straightforward, understandable user interface (hiding the more complex features). These simplifications should be done inside the security manager, so it is transparent for applications and devices in which setup they are deployed. Application developers should make no distinction between enterprise and small home.

A security configuration consists of two parts:

1. Certificates: certificates provide proof that an application is managed by a security manager. They can be used to gain access to resources of other peers or to provide resources themselves to others. The certificates describe the rights the holder has.
2. Policy: A policy is a list of Access Control Lists (ACLs). These ACLs describe how other peers can access the holder of the policy.

Security managers use AllJoyn to transfer this configuration to applications and devices they manage.

#### Security Manager Architecture

A security manager is a tool that can take multiple forms. For a home setup it can be a single application accessed by one person. For an enterprise setup, multiple administrators need to use it, so its core can run on a server, with some local application talking to it. When discussing the functional blocks of the security manager, it is important to understand that those blocks can reside on different machines and that for some of these we even have multiple instances.

* The manager provides certificates. In order to generate and sign certificates, it needs to have a certificate authority (CA).
* Configuration storage: The security manager should keep track of what the configuration looks like. To do so, it should persist the configuration data.
* UI: The administrator needs to interact with the security manager in order to make configuration changes. The user interfaces doesn't need to be part of the manager itself. It could be running in a web browser or it could offer a REST API, so that custom UI can be built on top.
* AllJoyn Agent (security manager agent): Configuration updates are sent using AllJoyn as the communication protocol. The agent is the component which does the interaction with the managed peers.

The following assumptions are made:

* The four functional blocks of the security manager can be combined into a single application, but it should be possible to run them in different applications or even on different hosts.
* A security manager can have multiple security manager agents acting on its behalf.
* The security manager topology is transparent for AllJoyn applications.
* A security manager is identified by the public key of its CA. We call this the key of the security manager.

The Alliance envisions multiple implementations of security managers and does not provide implementation specifications. The alliance does specify how security managers need to interact with AllJoyn security 2.0 based applications and devices.

* org.allseen.Security.PermissionMgmt.Notification: a session-less signal sent by the applications. This can be used to discover applications and devices.
* org.allseen.Security.PermissionMgmt: For changing the configuration of applications.

#### What the Security Manager manages

We already mentioned a number of times that a security manager manages applications and devices. But what does it mean and do we really manage applications and devices? The security manager agent will use AllJoyn security features to set up a secure connection to a peer. The only way it has to identify this peer is by the looking at that peer's public key. Since we hand out certificates granting rights to this key, in fact it means we are managing keys. So when asking what are we managing, we should ask ourselves who has access to a key? There is no easy answer to this questions. It all depends on the OS and platform the software is running on.

* On a plain Linux or Windows machine, applications can choose to protect data on a per-user basis, making it hard to protect the key from other applications running as the same user. On the other hand, the key is also not application-specific. When the same application runs as a different user, it can't access the key anymore.
* Operating systems on smartphone do a better job at sandboxing applications. The link between key and application is stronger there.

How many keys you need per device depends on the device:

* A single-function device (e.g., a temperature sensor) is considered as one big Application. One key to do all operations.
* Every app on a smartphone is considered as an app on its own, so one key per application.
* The built-in firmware of a smart TV is also considered as a single app. Applications installed on top of the firmware of the TV are separate apps and should have their own key.

##### What we can trust

The AllSeen alliance offers software stack that runs on top of some hardware within an OS. The stack can be embedded in an application which is installed on a device or could be integrated in a firmware of a device. The security manager cannot distinguish this. He only sees a remote peer. Furthermore the security manager cannot assume applications are running on trustworthy systems. If an application runs on a compromised or malicious system, there is little we can do inside the app to protect. A genuine application running on malicious system, should be treated as malicious. We should protect the ecosystem by:

1. Being able to revoke the rights granted to an application.
2. Make sure compromised or even malicious applications are limited to rights they were given. Since we can't trust the OS or hardware the applications is running on, these checks must be done at the remote peer side.

The protective measure should be defined so that a well-behaving app on a well behaving system can protect itself from any unwarranted access. If both peers are malicious, then there is little we can do. But then they don't need AllSeen to malicious things. There is a risk though that 2 malicious applications team up. Each individual app gets a small acceptable set of rights, but then combining their rights to launch an attack.

When claiming an application two considerations must be made:

1. Can I trust the application?
2. Can I trust the device and it is running on? But not only the device and it OS, but for desktops systems as well which other applications are there? These apps might to try get access to the keystore of the genuine app. This is not something we can fix within the AllSeen alliance. This remains an integration aspect.

##### Sharing Keystores

When an application is claimed, it will store its certificates inside a keystore. This keystore can be shared. The security manager nor the system can prevent applications from doing this. Is it recommended to share keystores? It has the advantage that you only have to claim one application, while multiple applications can use it. However the certificates in the store will only grant a limited set of permission to it users. Sharing the store only makes sense if it was granted all permissions required by its users. Sharing keystore can be allowed if the applications granted access to it are known upfront and the union of rights is known.

Sharing keystores does have some side effects, every app using the keystore will appear as a manageable application. The security will be able to manage one keystore via multiple apps. Adding an extra layer of complexity to get concurrent access right.

We also partially lose the ability to sandbox applications, as applications using a shared keystore get full set of rights linked to the store and not necessarily the ones they strictly need.

##### Applications integrated in firmware

The firmware of a device could consist of multiple smaller AllSeen applications. From end-user perspective you only want to claim this device once. Those applciations are allowed to share their keystore, but only of them should provide the PermissionMgmt interface. So only one application is seen from security manager perspective. When expressing the permission required for this application, it should request all permission required by the apps on that device.

##### Standalone applications

Standalone applications are apps downloaded and installed on a desktop computer, tablet or smart phone or something similar. Standalone applications should not share their keystore with other applications. If such an application is built out of separate sub-applications (each of them a using separate bus attachment), then they should follow the same rules as applications integrated in firmware.

#### Security Manager Operations

The security manager allows the following operations:

* security group management: create, update and delete security groups
  + allow grouping of applications. A group is uniquely defined by GUID and the public key of a security manager. Applications can become members of a group when they are issued a membership certificate for that group
* identity management: create, update and delete identities
  + Identities are used to define the users of application. Users can map to physical persons, but they could also a more conceptual meaning. An identity is represented by a GUID and the public key of the defining security manager. Applications can act on behalf of a user when they receive an identity certificate for that user. Applications should only have one identity certificate.
* application/key management:
  + Claim applications: make it managed by this security manager
  + manage AllJoyn certificates for these applications
  + manage policy (ACL's) of applications
  + force application to become un-managed again

#### Inter Security Manager Interaction

When applications interact with each other, they check if the interaction is allowed by their policies as previously set by their security manager. In practice, a peer must present a certificate (chain) signed by its security manager public key. Meaning that with the basic features we created silos, you can only talk to applications managed by your own security manager. In practice applications managed by different security managers need to interact with each other as well. We provide 2 ways to do this: Delegation and Group equivalence.

##### Delegation

###### Use case

I’m the administrator of my home ecosystem. I claim appliances in the home and provide them with configuration. I as administrator am the only person having access to the security manager. When my kids want to get access to an appliance, then they have to ask me to get approval for each application they want to use. This is not workable. With delegation, my security manager gives a membership certificate with delegation rights to the security manager of each of my kids. With this certificate, they can delegate these rights to their applications. They only need to ask one time and then they can make any of their applications part of my group. Even though my kids did not interact directly with each other, with these delegated certificates they interact with each other in the scope of this group.

###### Limitations

The followings are the limitations of using delegation.

* You can only authenticate members of the group. Mutual authenticated requests can only be done between members of the group.
* My kids get Remote-control rights for the TV by giving them a membership certificate with delegation rights for my TV Group. Their remote control applications become member of the TV group. If I give my TV a policy for the TV group, then the TV will allow the request from the RC apps of my kids. This requires my kids to define an ANY-USER policy for TV operations for their apps. This is ok for TVs RC operations. If mutual authentication is required, the TV must become member of the TV group as well.
* For a chat use case you need to know who is sending a message and to whom you’re sending messages to. So mutual authentication is required and all participants have to be in that group.
* As policy is defined on group level, it is impossible to differentiate between kids and parents within the scope of a single group. It would require separate groups to achieve this.

###### Delegating membership certificate Flow

In the X.509 membership certificate, the delegate concept is represented by the basicConstraints extension CA flag. If a grantee receives a membership certificate with the X.509 basicConstraints extension CA flag equal to true, the grantee can issue a membership certificate to others with the same ACL or more restrictive ACL. Any peer validating a certificate chain verifies that no further delegation has been done, or the chain is considered invalid.



Figure ‑8. Reissue membership certificate

##### Security Group Equivalence

Group equivalence tries to address the same problem as delegation but takes a different approach to solve it. With delegation, you give a certificate to an application. With that certificate the application can prove it is allowed access to a group. With group equivalence, we do the other way around. We define a policy on our managed applications that allows applications from a different security manager to get access. This would be as if we would pre-install the delegated membership certificate on all our managed applications. So when the peer comes around, he doesn't need to send the proof, the application already has it. Since policy comes from a trusted source, we don't need to distribute certificates, we can define a more compact ACL.

In practice, the security manager defines a group equivalence ACL for all applications that need to interact with the applications of the peer security manager. This ACL maps applications of that security manager into a local group. Those applications don’t need to be members of that group, but they need to prove that they are owned with an identity or membership certificate. If they can, the policy for the local group will be applied on them.

The group equivalence rule allows one-way authentication. For a mutual authenticated session, the peer security manager should install group equivalence for your security managers’ public key. The main advantage of group equivalence is that these rules do not require any knowledge of the groups owned by the peer security manager. In case of delegation, the receiver of membership certificates needs to know what the scope is of that group. Group equivalence is not transitive. If A declares group equivalence rules for B and C, that does not mean that a group equivalence exists between B and C. They have to explicitly install it themselves.

Group equivalence works very nice in a setup where every security manager represents a single user. Every person takes up the role of end-user as well and administrator. Then group equivalence easily allows you to express an ACL for another user. Users with similar rights can be mapped into the same group.

As example use case: Suppose we have a real-estate agent. When showing a house to clients, he'd like to show-off the AllSeen-enabled home automation system. This can be achieved with either delegation or group equivalence. The advantage of group equivalence is that if he potentially needs to show 100 homes, he can do it based on 1 certificate instead of 100 for the delegation scenario. There is less risk for information disclosure. If someone could get hold of the 100 certificates, then he can learn who the home sellers are. In the group equivalence case, the seller's public keys are in the policy of the agent's app and policy is never shared.

###### Implementing the security group equivalence concept

The security group equivalence concept can be implemented using the RESTRICTED\_USER peers. The friend’s identity certificate will be validated against the certificate authority’s public key entry in this RESTRICTE\_USER peer entry in the permission policy. These friend’s applications would have access to permissions assigned in the policy.



Figure ‑9. Add restricted user rules to an application

### Application Manifest

When considering where AllSeen enabled applications will run, smartphones are an obvious target. A lot of applications are available in various the app stores. Unfortunately not all of these applications are trustworthy. Example the flashlight app which asked for access to phonebook, network, etc. Same as the application is sandboxed on the smartphone, we would like to sandbox applications within the AllSeen security 2.0 context. If I install an AllSeen TV remote control app, then I would like it only to have rights to do TV operations. Nothing more. Since we can't trust the application, we can't assume it will behave properly. So these restrictions must be enforced by the peers. For the RC example, the TV must check the app has permissions. When remote control app tries to open the door, then the door must reject the call.

The main goal of an application manifest is to inform an admin which interfaces an application will produce and consume. Once the admin accepts the manifest, the manifest is signed and installed on the application. The signed manifest will be used to enforce the application cannot produce or consume any unwarranted interfaces.

A signed application manifest limits the potential interfaces a malicious application can access within a set of well-behaving applications.

The goal of an application manifest is similar application manifests on Android, in which an end-user has to accept a list of permissions when installing a new application on his phone which are enforced by Android. The implementation is however different, as described below.

#### Requirements

##### Manifest Format

The manifest must be expressed at the interface level. It may be expressed at the member level, but this is not recommended as this increases the complexity that needs to be handled by the admin.

##### Manifest Acceptance

The manifest must be presented to the admin in a user-friendly way. As the interface names might not be very informative, they must be mapped to a user-friendly description.

The descriptions of the interfaces must be trustworthy. As a malicious application can by definition not be trusted, the descriptions must be provided by a trusted third party.

The descriptions of the interfaces should be localized to the admin.

The AllSeen Alliance must provide descriptions for any standard AllSeen interface, as reviewed and accepted by the Interface Review Board (IRB).

The application developer must provide the descriptions of any application specific interfaces.

If a manifest is defined at the member level, a description for each listed member must be available.

##### Manifest Enforcement

When an application tries to produce or consume without a signed manifest granting him to do so, the AllSeen framework must return an authorization failure error.

The accepted manifest must be enforced by the peer application, as a malicious application may not be trusted to enforce it locally.

##### Manifest Update

Whenever an application is updated and does not require additional rights, it may still use the previously signed manifest. Only when the update requires additional rights, the admin must accept a new manifest for that application.

#### Implementation Scenario

When an administrator wants to add an application to one of his security groups, he needs to accept a manifest of the application. When he accepts the manifest, its contents digest will be encoded in a new identity certificate.

1. The security manager discovers the remote application through the NotifyConfig signal.
2. The admin adds the application to one of his security groups.
3. The security manager retrieves the manifest template of the application.
4. The security manager contacts a server to retrieve the human readable description of the interfaces and presents them to the admin.
5. The admin accepts (or rejects) the description of the manifest. When the admin rejects the manifest, the application will not receive a manifest.
6. The security manager encodes the digest of the requested (& accepted) permissions in a new identity certificate.
7. The security manager installs the new identity certificate and manifest on the application.

#### 

Figure 2‑10: Building Policy using manifest

##### Application

The application developer needs to embed the manifest in his application. There should be a platform specific callback function to retrieve the manifest that belongs to an application. For Android, it could be based on convention, providing the manifest as a file inside the application package. For small embedded devices, the manifest could be part of the application.

To ease the generation of membership certificates by the security manager, the manifest format should line up with the format that is used to express access rules in the membership certificates.

##### Interface Description Server

The server serving the descriptions of the interfaces can either be:

1. Hosted by the application developer for application specific interfaces. To prevent spoofing attacks, the URL of this server MUST be based on the reverse domain name of the interface name.
2. Hosted by the AllSeen Alliance for common AllSeen interfaces. This server MUST be hosted on a predefined URL. This server MUST be contacted over HTTPS.

##### Manifest Enforcement

As the manifests are encoded in the membership certificates, no additional enforcement mechanisms need to be implemented.

When applying security group specific policy rules, the remote peer will enforce the rules in the membership certificate, effectively enforcing the manifest of the application within that security group.

As no membership certificates can be applied for ANY-USER policy rules, manifests cannot be enforced for such policies.

## Access validation

### Validating policy on a producer

This is a typical producer validation of a consumer’s permissions when the consumer makes a method call on a secure interface.



Figure 2‑11. Validating policy on a producer

### Validating policy on a consumer

This is a typical consumer policy validation when the consumer application calls a secure method call.



Figure 2‑12. Validating policy for a consumer

### Validating policy on a consumer that requires producers membership in a security group

The following flow shows a policy enforcement on the consumer that requires the producer belong to a security group.



Figure 2‑13: Consumer policy requires producer belong to a security group

### Anonymous session

In scenarios when there is no trust established between two peers such as when a guest comes into the user's home, the guest’s consumer application can still control certain applications if and only if there are ACLs specified for ANONYMOUS\_USER installed on these devices.

Note that ANY\_USER refers to authenticated peers while ANONYMOUS\_USER refers to unauthenticated peers.

The consumer always use ECDHE\_ECDSA to contact a peer. If the key exchange fails, it can fallback to ECDHE\_NULL and contact the peer as an anonymous user. This process is automated so the application developer does not need to drive the key exchange process.



Figure 2‑14. Anonymous access

### Validating an admin



Figure 2‑15. Validating an admin

### Emitting a session-based signal

Before emitting a session-based signal to existing connections, the producer verifies whether it is allowed to emit the given signal to any authorized party. Upon receipt of the signal, the consumer checks whether it has the authorization to accept the given signal. The consumer verifies the producer’s manifest for proper authorization.



Figure 2‑16. Validating a session-based signal

## Authorization data format

### The format is binary and exchanged between peers using AllJoyn marshalling

The authorization data will be in binary format. The following guidelines are used for exchanging and persisting the authorization data:

1. The authorization data will use AllJoyn marshalling to exchange with other peers.
2. The AllJoyn marshalling will be used to generate buffers to be signed.
3. The AllJoyn marshalling will be used to serialize the data for persistence.
4. The parser will ignore any field that it does not support.

### Format Structure

The following diagram describes the format structure of the ACL data.



Figure 2‑17: Authorization Data Format Structure

#### Authorization data field definition

Root level

| Name | Data type | Required | Description |
| --- | --- | --- | --- |
| version | number | yes | The specification version number. The current spec version number is 1. |
| serialNumber | number | yes | The serial number of the policy. The serial number is used to detect of an update to an older policy. |
| ACLs | Array of ACLs | yes | List of access control lists. |

Access Control List

| Name | Data type | Required | Description |
| --- | --- | --- | --- |
| peers | array of objects | no | List of peers. There are multiple types of peers. A peer object has the following fields:   |  |  |  |  | | --- | --- | --- | --- | | **Name** | **Data**  **Type** | **Required** | **Description** | | type | number | yes | The peer type. The followings are the valid type of peers:   * ANONYMOUS\_USER (unauthenticated peer) * ANY\_USER (authenticated peer) * RESTRICTED\_USER (authenticated user validated by a specific certificate authority) * PUBLIC\_KEY * SECURITY\_GROUP | | publicKey | Public Key | no | The peer key info data. Depending on peer type, the publicKey is:   * ANONYMOUS\_USER – not applicable * ANY\_USER – not applicable * RESTRICTED\_USER – the public key of the certificate authority * PUBLIC\_KEY – the public key of the peer * SECURITY\_GROUP – the public key of the security group authority | | sgID | GUID | No | Security group ID. This is applicable only the type SECURITY\_GROUP. | |
| rules | array of rules | no | List of allowed rules. The application is allowed to perform the actions specified in the given rules.  The default rule is to allow nothing. |

Rule Record

| Name | Data type | Required | List of values | Description |
| --- | --- | --- | --- | --- |
| obj | string | no |  | Object path of the secured object. A \* at the end indicates a prefix match. When there is no \*, it is an exact match. |
| ifn | string | no |  | Interface name. A \* at the end indicates a prefix match. When there is no \*, it is an exact match. |

Interface Member Record

| Name | Data type | Required | List of values | Description |
| --- | --- | --- | --- | --- |
| mbr | string | no |  | Member name. A \* at the end indicates a prefix match. When there is no \*, it is an exact match. |
| type | number | no | * 0: any * 1: method call * 2: signal * 3: property | Message type.  Default is any |
| action | byte | no |  | The action mask flag. The list of valid masks:   * 0x01: Denied * 0x02: Provide – allows sending signal, exposing method calls and producing properties * 0x04: Observe – allows receiving signals and getting properties * 0x08: Modify – set properties and make method calls |

#### Enforcing the rules at message creation or receipt

The following table lists the required action mask base on the message.

Table -2: Action Mask Matrix

|  |  |  |
| --- | --- | --- |
| **Message Action** | **Local Policy** | **Remote peer’s manifest** |
| send GetProperty | Remote peer has PROVIDE permission for this property | Remote peer has PROVIDE permission for this property |
| receive GetProperty | Remote peer has OBSERVE permission for this property | Remote peer has OBSERVE permission for this property |
| send SetProperty | Remote peer has PROVIDE permission for this property | Remote peer has PROVIDE permission for this property |
| receive SetProperty | Remote peer has MODIFY permission for this property | Remote peer has MODIFY permission for this property |
| send method call | Remote peer has PROVIDE permission for this method call | Remote peer has PROVIDE permission for this method call |
| receive method call | Remote peer has MODIFY permission for this method call | Remote peer has MODIFY permission for this method call |
| send signal | Remote peer has OBSERVE permission for this signal | Remote peer has OBSERVE permission for this signal |
| receive signal | Remote peer has PROVIDE permission for this signal | Remote peer has PROVIDE permission for this signal |

##### Initial policy after claim

Right after the application is claimed, an initial policy is created automatically with the following feature:

1. Admin group has full access
2. Outgoing messages are authorized
3. Incoming messages are not authorized

An example of the initial policy is:

peer: SECURITY\_GROUP

pubKey: admin authority key

sgID: admin group ID

ifn: \*

mbr: \*

action: 0x0E (PROVIDE | OBSERVE | MODIFY)

peer: ANY\_USER

ifn: \*

mbr: \*

action: 0x06 (PROVIDE | OBSERVE)

#### Search Algorithm

Whenever an encrypted message is created or received, the authorization rules are searched using the message header data (object path, interface name, and member name) and the requested permission listed in Table 2-2: Action Mask Matrix.

#### Matching Algorithm within an ACL

The following matching algorithm is used to find a match within an ACL. Once a match or an explicit deny is found within the rules, the search stops.

* If the rule specifies both object path and interface name, the message must match both.
* If the rule specifies object path, the message must match the object path.
* If the rule specifies interface name, the message must match the interface name.
* If the rule specifies member name, the message must match the member name.
* If the rule specifies a type, the message must match the type
* Verify whether the requested permission is allowed by the action mask at the member.
  + When a member name has an exact match and the action mask indicates access is explicitly denied then the explicit deny is found.
  + When a member name has an exact match and is allowed by the action mask then a match is found.
  + When a member name has a prefix match and the action mask indicates access explicitly denied then the explicit deny is found.
  + When a member name has a prefix match and is allowed by the action mask then is a match is found.

#### Search Priorities for Policy ACLs

Policy ACLs are searched in this order. Once a match or an explicit deny is found, the search stops.

1. All public key policy ACLs
2. All security group policy ACLs are applied in undefined order. Per security-group-in-common, the materialized authorization rules are the intersection of the authorization rules between the consumer and producer. A match found in any of the security group policy ACLs is consider a match is found.
3. The any-user policy ACLs.
4. The anonymous policy ACLs

## Certificates

The following subsections detail the supported certificates. The certificate format is X.509 v3. The certificate lifetime will be considered in order to avoid having to revoke the certificate. However, certain devices do not have access to a trusted real time clock. In such cases, applications on those devices will not be able to validate the certificate lifetime, thus relying on certificate revocation.

### Main Certificate Structure

All AllSeen X.509 certificates have the following ASN.1 structure. Currently only the ECDSA (prime256v1) certificates are supported.

Certificate ::= SEQUENCE {

tbsCertificate TBSCertificate,

signatureAlgorithm SEQUENCE { 1.2.840.10045.4.3.2 (ecdsa-with-sha256) },

signatureValue BIT STRING

}

TBSCertificate ::= SEQUENCE {

version v3(2),

serialNumber INTEGER,

signature SEQUENCE { 1.2.840.10045.4.3.2 (ecdsa-with-sha256) },

issuer SEQUENCE { 2.5.4.3 (commonName), UTF8 STRING },

validity Validity,

subject Name,

subjectPublicKeyInfo SEQUENCE { 1.2.840.10045.2.1 (id-ecPublicKey), 1.2.840.10045.3.1.7 (prime256v1), BIT STRING },

issuerUniqueID IMPLICIT UniqueIdentifier OPTIONAL,

subjectUniqueID IMPLICIT UniqueIdentifier OPTIONAL,

extensions EXPLICIT

}

#### Security 2.0 Custom OIDs

All Security 2.0 custom OIDs will start with 1.3.6.1.4.1.44924.1 where 1.3.6.1.4.1.44924 is the registered AllSeen Alliance Private Enterprise Number.

### Identity certificate

The identity certificate is used to associate application, user or device with an identity alias. This allows an identity alias to have a number of identity certificates installed in different keystores.

The identity alias is encoded in the SubjectAltName field in the extensions.

The extensions include the following fields:

* CertificateType: the type of certificate within the AllSeen ecosystem. An identity certificate has certificate type equal to 1.
* SubjectAltName: the alias for the identity.
* AssociatedDigest: the digest of the associated manifest data.

Both the CertificateType and AssociatedDigest have custom OIDs under the Security 2.0 root.

SubjectName ::= SEQUENCE { 2.5.4.3 (commonName), UTF8 STRING },

Extensions ::= SEQUENCE {

BasicConstraints SEQUENCE { 2.5.29.19 (basicConstraints), BOOLEAN (FALSE) },

CertificateType SEQUENCE { 1.3.6.1.4.1.44924.1.1 (AllSeen Certificate Type), INTEGER (1) },

SubjectAltName SEQUENCE { 2.5.29.17 (subjectAltName), OCTET STRING },

AssociatedDigest SEQUENCE { 1.3.6.1.4.1.44924.1.2 (AllSeen Certificate Digest), 2.16.840.1.101.3.4.2.1 (sha-256), OCTET STRING }

}

### Membership certificate

The membership certificate is used to assert an application, user or device is part of a security group.

The security group identifier is encoded with a 16 network byte order octets encoded in the SubjectAltName field in the extensions.

The extensions include the following fields:

* CertificateType: the type of certificate within the AllSeen ecosystem. A membership certificate has certificate type equal to 2.
* SubjectAltName: the security group ID.

The CertificateType has a custom OID defined under the Security 2.0 root.

SubjectName ::= SEQUENCE { 2.5.4.11 (organizationalUnitName), UTF8 STRING, 2.5.4.3 (commonName), UTF8 STRING },

Extensions ::= SEQUENCE {

BasicConstraints SEQUENCE { 2.5.29.19 (basicConstraints), BOOLEAN default FALSE },

CertificateType SEQUENCE { 1.3.6.1.4.1.44924.1.1 (AllSeen Certificate Type), INTEGER (2) },

SubjectAltName SEQUENCE { 2.5.29.17 (subjectAltName), OCTET STRING }

}

## Sample use cases

The solution listed here for the use cases is just a typical solution. It is not intended to be the only solution.

### Users and devices

Users: Dad, Mom, and son

| Security Group | Members |
| --- | --- |
| homeAdmin | Dad, Mom |
| sonAdmin | Son |
| dadOnlyAdmin | Dad |
| livingRoom | TV, living room tablet, son’s room TV, master bedroom TV, master bedroom tablet |
| masterBedrom | Master bedroom tablet |

| Room | Devices | Notes |
| --- | --- | --- |
| Living room | TV, Set-top box, tablet, Network-attached Storage (NAS) | * All devices claimed by Dad and managed by Mon and Dad using the security group homeAdmin * All devices are accessible for the whole family |
| Son’s bedroom | TV | * Claimed and managed by son * TV is allowed to interact with living room devices for streaming data |
| Master bedroom | TV, tablet | * TV used by Mom and Dad only * Tablet used by Dad only * TV is allowed to interact with living room devices for streaming data * Tablet has full control of living room devices including the parent control feature |

### Users set up by Dad



Figure 2‑18. Use case - users set up by Dad

### Living room set up by Dad



Figure 2‑19. Use case - living room set up by Dad

### Son's bedroom set up by son



Figure 2‑20. Use case - son's bedroom set up by son

### Master bedroom set up by Dad



Figure 2‑21. Use case - master bedroom set up by Dad

### Son can control different TVs in the house



Figure 2‑22. Use case – Son can control different TVs in the house

### Living room tablet controls TVs in the house



Figure 2‑23. Use case - Living room tablet controls TVs

# Enhancements to Existing Framework

## Crypto Agility Exchange

In order to provide the AllJoyn peers to express the desire to pick some particular cryptographic cypher suite to use in the key exchange and the encryption of the messages, new key exchange suite identifiers will be added to the framework to express the choice of cipher and MAC algorithms. The new identifiers may come from the list of TSL cipher suites specified in [Appendix A.5 of TLS RFC5246](http://tools.ietf.org/html/rfc5246#page-75) , [RFC6655](http://tools.ietf.org/html/rfc6655), and [RFC7251](http://tools.ietf.org/html/rfc7251).

The following table shows the list of existing key exchange suites:

|  |  |  |
| --- | --- | --- |
| **AllJoyn Key Exchange Suite** | **Crypto Parameters** | **Availability** |
| ALLJOYN\_ECDHE\_NULL | * Curve NIST P-256 (secp256r1) * AES\_128\_CCM\_8 * SHA256 | * Standard Client * Thin Client |
| ALLJOYN\_ECDHE\_PSK | * Curve NIST P-256 (secp256r1) * AES\_128\_CCM\_8 * SHA256 | * Standard Client * Thin Client |
| ALLJOYN\_ECDHE\_ECDSA | * Curve NIST P-256 (secp256r1) * AES\_128\_CCM\_8 * SHA256 * X.509 certificate | * Standard Client * Thin Client |
| ALLJOYN\_RSA\_KEYX | * AES\_128\_CCM\_8 * SHA256 * X.509 certificate | * Standard Client |
| ALLJOYN\_PIN\_KEYX | * AES\_128\_CCM\_8 | * Standard Client version 14.12 or older * Thin Client version 14.02 or older |
| ALLJOYN\_SRP\_KEYX | * AES\_128\_CCM\_8 | * Standard Client |
| ALLJOYN\_SRP\_LOGON | * AES\_128\_CCM\_8 | * Standard Client |

The following table shows the potential list of TLS cipher suites to be supported. Other suites will be added as codes are available.

|  |  |  |  |
| --- | --- | --- | --- |
| **TLS cipher suite** | **Additional Crypto Parameters** | **Availability** | **RFC** |
| TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8 | * Curve NIST P-256 (secp256r1) * SHA256 * X.509 certificate | * Standard Client * Thin Client | [7251](http://tools.ietf.org/html/rfc7251) |
| TLS\_RSA\_WITH\_AES\_128\_CCM\_8 | * SHA256 * X.509 certificate | * Standard Client | [6655](http://tools.ietf.org/html/rfc6655) |

## Permission NotifyConfig Announcement

The Permission module provides a session-less signal to allow the Security Manager discovering the applications to claim or to distribute updated policy or certificates. The current features provided by the About session-less signal does not fulfill the Security Manager discovery requirement. The signal provides the following information:

1. A number field named **claimable** to show the claim state of the application. The possible values of this field are:
   * + 0 -- not claimable
     + 1 – claimable
     + 2 - claimed
2. The authentication GUID and public key
3. The permission policy serial number

This signal is emitted when

1. The bus attachment is enabled with peer security using ECDHE key exchanges
2. The application is claimed or do a factory reset
3. The application has a permission policy installed
4. The application has its permission policy removed

# Features In Future Releases

### Certificate revocation (not fully designed)

The application will validate the certificate using a revocation service provided by the Security Manager. The revocation service is a distributed service.

The Certificate Revocation Service is expected to provide a method call that takes in the certificate and return whether the given certificate is revoked.

The application looks in the “self” section of its installed policy for the peer that provides the Certificate Revocation Service. If the application can’t locate any of the Certificate Revocation Service, the certificate revocation check will be skipped.

If a membership certificate is revoked, all signed authorization data related to the membership certificate is no longer valid.

### Distribution of policy updates and membership certificates (not fully designed)

The Distribution Service is a service provided by a Security Manager. This service provides persistent storage and high availability to distribute updates to applications.

An admin uses the Security Manager to generate updated policy and membership certificates, encrypt the payload with a session key derived from a nonce value and the master secret for the <sender, recipient> pair. The package including the sender public key, recipient public key, nonce, and encrypted payload is sent to the Distribution Service to delivery to the recipient. The recipient uses the information in the package to locate the master secret to generate the corresponding session key to decrypt the payload. Once the decryption is successful, the recipient signs the hash of the package and provide the signature in the reply.



Figure 4‑1. Distribution of policy update and certificates

### Policy Templates

An application developer can define policy templates to help the Security Manager to build consumer and producer policies. A policy template provides the following data in:

* Specification version number
* List of permission rules

# Future Considerations

## Broadcast signals and multipoint sessions

All security enhancements for broadcast signals and multipoint sessions will be considered in future releases of Security 2.0.