AllJoyn™ Security 2.0 Feature

High-Level Design Document

Rev 1 Update 0

August 8, 2014

Contents

1 Introduction 5

1.1 Purpose and scope 5

1.2 Revision history 5

1.3 Acronyms and terms 5

2 System Design 7

2.1 Overview 7

2.2 Premises 8

2.3 Typical operations 10

2.3.1 Claim a factory-reset device 10

2.3.2 Define a guild 12

2.3.3 Example of building a policy 12

2.3.4 Install an ANY-USER policy 12

2.3.5 Install a guild-specific policy 13

2.3.6 Add an application to a guild 14

2.3.7 Add a user to a guild 15

2.3.8 Delegating membership certificate 16

2.3.9 Add a guild equivalence certificate to an application 17

2.3.10 Access revocation (Work-In-Progress) 18

2.3.11 Distribution of policy and membership certificate updates (Work-in-progress) 19

2.4 Access validation 20

2.4.1 Validation flow 20

2.4.2 Validating a consumer policy 21

2.4.3 Exchanging a trust profile during session establishment 22

2.4.4 Anonymous session 24

2.4.5 Validating an admin user 25

2.4.6 Emitting a session-based signal 25

2.5 Authorization data format 26

2.5.1 Format 26

2.5.2 Examples 30

2.5.3 Policy Templates (Work-In-Progress) 34

2.6 Certificates 34

2.6.1 Policy certificate 34

2.6.2 Membership certificate 35

2.6.3 User equivalence certificate 35

2.6.4 Identity certificate 36

2.6.5 Guild equivalence certificate 36

2.7 Sample use cases 37

2.7.1 Users and devices 37

2.7.2 Users set up by Dad 38

2.7.3 Living room set up by Dad 39

2.7.4 Son's bedroom set up by son 40

2.7.5 Master bedroom set up by Dad 41

2.7.6 Son can control different TVs in the house 42

2.7.7 Living room tablet controls TVs in the house 43

3 Future Considerations 44

3.1 Broadcast signals and multipoint sessions 44

Figures

Figure 2‑1. Security system diagram 8

Figure 2‑2. Claim a factory-reset device without out-of-band registration data 11

Figure 2‑3. Claiming a factory-reset device using out-of-band registration data 12

Figure 2‑4. Install an ANY-USER policy 13

Figure 2‑5. Install a guild-specific policy 14

Figure 2‑6. Add an application to a guild 15

Figure 2‑7. Add a user to a guild 16

Figure 2‑8. Distribution of policy/membership certificate updates 20

Figure 2‑9. Validation Flow 21

Figure 2‑10. Validating a consumer policy 22

Figure 2‑11. Exchange a trust profile 23

Figure 2‑12. Anonymous access 24

Figure 2‑13. Validating an admin user 25

Figure 2‑14. Validating a session-based signal 26

Figure 2‑15. Use case - users set up by Dad 38

Figure 2‑16. Use case - living room set up by Dad 39

Figure 2‑17. Use case - son's bedroom set up by son 40

Figure 2‑18. Use case - master bedroom set up by Dad 41

Figure 2‑19. Use case – Son can control different TVs in the house 42

Figure 2‑20. Use case - Living room tablet controls TVs 43

Tables

Table 2‑1. Security 2.0 premises 8

Table 2‑2. Policy certificate fields 34

Table 2‑3. Guild-specific certificate fields 35

Table 2‑4. User equivalence certificate fields 35

Table 2‑5. Identity certificate fields 36

Table 2‑6. Guild equivalence certificate fields 37

# 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 |

## 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. |
| Provider | An AllJoyn application advertises its interfaces so other AllJoyn application may access/control it. |
| Consumer | An AllJoyn application which is able to control or uses services provided by another AllJoyn application. |
| 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. |
| AllJoyn framework | Open source peer-to-peer framework that allows for abstraction of low-level network concepts and APIs. |
| 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. |
| User | The person or business entity interacting with AllJoyn applications. |
| Factory-reset device | A device 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 |
| Guild | A logical grouping of devices, applications, and users. It is identified by a guild ID which is a GUID. An application can be installed with a policy to expose services to members of the guild. An application or user holding a membership certificate is in fact a member of the guild. Any member of the guild can access the services exposed to the guild by the applications with policies defined for that guild. |
| Guild Authority | A guild authority is the user or application that defines the guild policy and grant membership certificates to other. The guild authority is the certificate authority for that guild. |
| Holder | The application or user possessing a certificate. |
| Issuer | The application or user signing a certificate. |
| OOB | Out Of Band |
| Permission Management module | The AllJoyn Core module that handles all the permission authorization. |
| PermissionMgmt | A set of AllJoyn interfaces to manage the permissions for the AllJoyn application. The implementation is provided by the Permission Management module |
| Security Manager | A set of AllJoyn interfaces to manage cryptographic keys, generate and distribute certificates. |
| Security Appliance | A security appliance is a type of Security Manager that is always present. |
| IoE | Internet of Everything |
| Peer | Application participating in the AllJoyn messaging. |
| SHA-256 | Secure Hash Algorithm SHA-2 with digest size of 256 bits or 32 bytes. |
| Trust profile | Information used by peers to introduce themselves when contacting each other. |
| Certificate Authority (CA) | Entity that issues a digital certificate |

# System Design

## Overview

The goal of the Security 2.0 feature is to allow an application to validate access to secure interfaces or secure 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 authorization before any message action can be taken.

The Security Manager is optional service that helps the user with key management and permission rules building. Using policy templates defined by application developer, the Security Manager builds the application manifest to let the end-user authorize which interactions the application can do.

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 ‑. 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 identification | All peers are identified by a cryptographic public key |
| Admin | An admin (or administrator) is a peer 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 add/remove another admin |
| Claiming | Incorporate a factory-reset device with the Permission Management | * A factory-reset device has no list of certificate authorities. * A factory-reset device has no admin * Anyone can claim as an admin for a factory-reset device. |
| 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 peer if the policy is signed by an admin of the given application. * Developers can define policy templates to help the user with policy building. * Guild-specific policy specifies the permissions granted to members of the guild. The guild authority becomes a certificate authority for that particular guild. * A policy may exist at the provider or consumer side. Policy enforcement applies wherever it resides. * A policy is considered private. It is not exchanged with any peer. * An application may zero or more policies. |
| Membership certificate | A membership certificate is the proof of a guild membership | * Membership certificates are exchanged between peers. The authorization data signed by this certificate are used for mutual authorization purposes. * An application trusts the membership certificate if the issuer or any subject in the issuer’s certificate chain matches any of the application certificate authorities. * A membership certificate holder can generate additional membership certificate for the given guild 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 guild ID. * A device or application can accept any number of membership certificates |
| User equivalent certificate | A user equivalence certificate allows the holder to act like the issuer | The holder has the same access rights as the issuer |
| Authorization data | The permission rules | * Authorization data are not present in the membership or the policy certificate * The certificate holds the digest of the authorization data. * Authorization data can be requested from the certificate holder. |
| Guild equivalence certificate | Certificate maps other guilds to a specific guild | * An admin can add a guild equivalence certificate to the application. This mechanism allows other guilds to map to a specific guild. * The subject in the certificate is the equivalence guild authority’s public key. A membership certificate generated from that guild authority or its delegates will have access to the specific guild defined in the guild equivalence cert. |
| Identity certificate | Certificate that signs the identity information and optional vCard data. | * Certificate with a digest of the actual identity data. * The Certificate has an alias field for that identity in addition to the external Identification data * A peer can request for the other peer’s identity certificate and identity data. * An application trusts identity certificate issued by any of the application’s certificate authorities and guild equivalence authorities. * An application may have one or more identity certificates. |

## Typical operations

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

### Claim a factory-reset device

A user can claim any factory reset device during the claiming interval. Claiming is first-come, first-claim action. That user becomes the admin. If there is no claimant during the claiming interval, the device becomes unclaimable. The procedure to make the device to become claimable again is manufacturer’s specific.

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



Figure ‑. Claim a factory-reset device without out-of-band registration data

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

A device manufacturer can provision a key to support the claiming process. 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 establish connection with the device.



Figure ‑. Claiming a factory-reset device using out-of-band registration data

### Define a guild

A user can define a guild (logical grouping of devices and users) using the Security Manager. When the user specifies a guild name, the Security Manager creates the guild ID (typically a GUID value).

### Example of building a policy

A user uses a Security Manager application to build a policy. The application queries the AllJoyn About feature data and the list of policy templates from the device. The Security Manager application can do further introspection of the device for the detailed information of secured interfaces and secured objects, and prompts the user to select the permissions to include in the policy.

### Install an ANY-USER policy

An admin can install an ANY-USER policy for the application. This policy specifies all the authorizations for any user.



Figure ‑. Install an ANY-USER policy

### Install a guild-specific policy

An admin can install a guild-specific policy on the application. This policy specifies how the given application behaves based on its role as a provider and/or a consumer. See Authorization data format for more details on the format of the authorization data. Typically, a provider has policies installed, but occasionally a consumer may also have a policy. For a provider, the policy specifies all the authorizations to any consumer presenting a guild membership certificate. For a consumer, the policy specifies whether it has the privilege to send a command to the provider or to receive a signal from the provider.

Installing a guild-specific policy does not allow the holding application to access other applications in the guild.



Figure ‑. Install a guild-specific policy

### Add an application to a guild

An admin signs a membership certificate with the given guild ID and installs it in the application. This act adds the application to the guild. In order for a provider to emit signals to other members of the guild, the provider must have a membership certificate with proper authorization to do so.



Figure ‑. Add an application to a guild

### Add a user to a guild

The guild authority uses the Security Manager to generate the membership certificate for the user for the given guild ID. The guild authority can restrict the permissions for this user.



Figure ‑. Add a user to a guild

### Delegating membership certificate

If a grantee receives a membership certificate with a delegate flag enabled, the grantee can issue the same membership certificate to others with the same authorization or more restrictive authorization. The peer verifies that no further delegation is allowed.



Figure 2‑8. Reissue membership certificate

### Add a guild equivalence certificate to an application

An admin can add a guild equivalence certificate to the application so the membership certificates issued by other certificate authorities (like friends) can be trusted. These certificate holders would only have access to permissions assigned to that specific guild.



Figure ‑9. Add a guild equivalence certificate to an application

### Access revocation (Work-In-Progress)

The following are proposed ideas to handle the revoking access action. Pros and cons are included with each proposal.

#### Proposal 1 – periodic updates

This access revocation method implements a periodic update of policies in applications and reissuing new membership certificates.

| Pros | Cons |
| --- | --- |
| No need to revoke any existing membership certificates already handed out to users. | * Requires automation since the manual process will be tedious. * Requires additional propagation if a certificate allowed additional delegation. * No immediate revocation |

#### Proposal 2 – short black list and periodic updates

This access revocation method implements the following:

* A periodic update of policies in applications and reissuing new membership certificates.
* During the update, a short black list can be broadcasted to all applications to cache. Membership certificates are verified against this black list.

| Pros | Cons |
| --- | --- |
| No need to revoke any existing membership certificates already handed out to users | Requires automation since the manual process will be tedious |
| Access revocation can be in effect sooner because of black list check | Requires more resources to store the black list |

### Distribution of policy and membership certificate updates (Work-in-progress)

An admin uses the Security Manager to generate updated policy and membership certificates and deliver them to the Security Appliance for distributions to the applications he/she owns. These certificates are signed by an admin, and the subject is the specific application. As the result, the application will trust the certificate and its data (policy or guild membership).

The Security Appliance broadcasts that updates are available so the applications can connect the Security Appliance to retrieve the updated certificates. The Security Appliance attempts to install the updated certificates if any application does not retrieve its updated certificates after a certain time period.

A typical notification message from the Security Appliance contains the issuer’s public key, peerID or guild ID, and version information.



Figure ‑. Distribution of policy/membership certificate updates

## Access validation

### Validation flow

A typical provider validation of the consumer permissions when a secure interface is requested.



Figure ‑. Validation Flow

### Validating a consumer policy

A typical consumer policy validation when a secure method call is called by the consumer’s app.



Figure ‑. Validating a consumer policy

### Exchanging a trust profile during session establishment

During the AllJoyn session establishment, the peers exchange the trust profile to determine what they have in common.

* A consumer’s trust profile typically holds the list of membership guilds (guild ID and membership cert signer chain), at least one certificate (typically it’s the identity certificate) with the issuer’s certificate chain attached.
* A provider’s trust profile typically holds the list of policy guilds (guild ID and guild-authority), membership guilds (guild ID and membership cert signer chain), at least one certificate (typically it’s the identity certificate) with the issuer’s certificate chain attached.

This process is initiated by the session joiner side (typically it’s the consumer side). Based on the trust profile data received, the consumer determines which membership certificates to send to the provider. The consumer also determines which membership certificates it requests from the provider. The access data are cached on both sides to enforce the authorization rules as the message comes in.



Figure ‑. Exchange a trust profile

### 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 devices if and only if there is an ANY\_USER policy installed on these devices. In such a scenario, the consumer application can ask the Permission Management module to switch to an ECDHE\_NULL session for a short period of time.



Figure ‑. Anonymous access

### Validating an admin user



Figure ‑. Validating an admin user

### Emitting a session-based signal

Before emitting a session-based signal to existing connections, the provider verifies whether it is allowed to emit the given signal to the guild members. The provider also verifies that the recipient is authorized to receive the signal. Upon receipt of the signal, the consumer checks whether it has a policy to allow it to accept the given signal. The consumer also checks whether the provider is authorized to emit the given signal.



Figure ‑. Validating a session-based signal

## Authorization data format

The authorization data format is in JSON format.

### Format

{

"admin": [

{

"version": 1,

"peer": [{"type": "DSA", "ID": "ownerPubKey"}]

}

],

"provider": [

{

"version": number,

"peer":

[{"type": "typeValue", "ID": "peerID", "authority":"key"}],

"allow": [rules],

"deny": [rules]

}

],

"consumer": [

{

"version": number,

"peer":

[{"type": "typeValue", "ID": "peerID", "authority":"key"}],

"deny": [rules],

"allow": [rules]

}

]

}

A rule is defined as

{

"obj": "object path" | "prefix\*" ,

"ifn": "interface name" | "prefix\*" | "\*",

"mbr": "member name",

"type": "M" | "S" | "P"

"read-only": boolean

}

#### Authorization data field definition

Root level

| Name | Data type | Required | Description |
| --- | --- | --- | --- |
| admin | Array of Policy objects | no | List of admin policies. An admin policy specifies the list of peers who have the admin privilege for the application. An admin peer becomes a certificate authority for the application. |
| provider | Array of Policy objects | no | List of provider policies. A provider policy specifies the features the application can provide to others. |
| consumer | Array of ACL objects | no | List of consumer ACLs. A consumer ACL specifies the features the application can invoke on others. |

Policy/ACL

| Name | Data type | Required | Description |
| --- | --- | --- | --- |
| version | Number | no | Version number of the authorization data. Can be used for notification of updates. |
| peer | 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 | string | yes | The peer type. The followings are the valid type of peers:   * ANY * PSK * DSA * GUILD | | ID | string | no | The peer ID. Depending on peer type, the ID is:   * ANY – not applicable * PSK – the PSK name * DSA – the public key * GUILD – the GUID of the guild | | authority | string | no | The guild authority. This field is applicable for a guild.  The guild authority becomes a certificate authority for the given guild. | | psk | string | no | The hex encoded shared secret. It is applicable for peer type PSK. | |
| allow | 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. |
| deny | array of rules | no | A short cut to say allow all except these rules. This field is ignored if the allow field is present and different than “\*” |

Rule used in Deny or Allow record

| Name | Data type | Required | List of values | Description |
| --- | --- | --- | --- | --- |
| obj | string | no |  | Object path of the secured object. A \* indicates a prefix match.  **If the object path is specified, the remaining fields are ignored. In other words, a rule is either object path specific or interface specific.** |
| ifn | string | no |  | Interface name. A \* indicates a prefix match. |
| mbr | string | no |  | Member name |
| type | string | no | * M: method call * S: signal * P: property | Message type. If the type is not specified, the Interface definition will be examined in the following order to determine whether the member name is.   1. A method call or signal. 2. A property. |
| read-only | boolean | no |  | Read-only flag applicable to Property only. The default value is false. |
| mutualAuth | boolean | no |  | Mutual authorization required. Both peers (local and remote) are required to be granted.  Default is yes for signal and no for method call and property. |

#### Mapping between message permission and policy permission

The following tables describe the mapping between the message action (send or receive) and the policy permission.

Permission required for Message action

|  | Provider | | Consumer | |
| --- | --- | --- | --- | --- |
| Member type |  | read-only=false |  | Read-only=false |
| property | receive GetProperty | receive SetProperty | send GetProperty | send SetProperty |
| method call | receive |  | send |  |
| signal | send |  | receive |  |

Enforcing the rules at message creation or receipt

| Message action | Required permission | | Affected member |
| --- | --- | --- | --- |
| Local peer | Remote peer |
| send GetProperty | Consumer | Provider (if mutual authorization is required) | property |
| receive GetProperty | Provider | Consumer | property |
| send SetProperty | Consumer read-only=false | Provider read-only=false (if mutual authorization is required) | property |
| receive SetProperty | Provider | Consumer read-only=false | property |
| send method call | Consumer | Provider (if mutual authorization is required) | method call |
| receive method call | Provider | Consumer | method call |
| send signal | Provider | Consumer | signal |
| receive signal | Consumer | Provider | signal |

#### Search order

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).

The following search order is performed against the authorization rules to find a match. A rule without wild card is stronger than one with wild card. Once a match is found, the search stops.

1. Object path
2. Interface name, member name.
3. Interface name.

#### Materializing the rules

Policies are applied to a peer connection as follows:

1. The ANY-USER policy is applied
2. All guild-in-common policies are applied in undefined order. Per guild-in-common, the materialized authorization rules are the intersection of the authorization rules between the consumer and provider. This intersection will be unioned with the previous result.
3. All peer-specific policies are applied in undefined order. The additional materialized authorization rules will be unioned with the previous result.

### Examples

The interface names and members are referenced from the following sample introspection XML.

<node

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="https://www.allseenalliance.org/schemas/introspect.xsd">

<interface name="org.allseenalliance.control.OnOff">

<method name="On"></method>

<method name="Off"></method>

</interface>

<interface name="org.allseenalliance.control.TV">

<property name="Channel" type="u" access="readwrite"></property>

<method name="Up"></method>

<method name="Down"></method>

<signal name="ChannelChanged">

<arg name="newChannel" type="u"/>

</signal>

</interface>

<interface name="org.allseenalliance.control.Mouse">

<property name="MousePosition" type="(ii)" access="readwrite"/>

<method name="ClickMouse"></method>

<method name="WheelMouse">

<arg name="direction" type="q" direction="in"/>

<arg name="newMousePosition" type="(ii)" direction="out"/>

</method>

</interface>

<interface name="org.allseenalliance.control.ParentalControl">

<method name="RateChannel">

<arg name="channel" type="u" direction="in"/>

<arg name="rating" type="q" direction="in"/>

</method>

<method name="EnableChannel">

<arg name="channel" type="u" direction="in"/>

</method>

<method name="DisableChannel">

<arg name="channel" type="u" direction="in"/>

</method>

</interface>

</node>

#### Sample authorization data in a provider policy file

This sample shows a provider authorization data containing an “any-peer” policy and some guild-specific policies.

{

"provider": [

{ "version": 1,

"peer": [{"type": "ANY"}],

"allow": [

{"ifn": "org.allseenalliance.control.OnOff"}

]

},

{ "version": 1,

"peer": [{"type": "GUILD", "ID": "LivingRoomGuildGUID", "authority": "guildPubKey"}],

"allow": [

{"ifn": "org.allseenalliance.control.TV", "mbr": "Up"},

{"ifn": "org.allseenalliance.control.TV", "mbr": "Down"},

{"ifn": "org.allseenalliance.control.TV", "mbr": "Channel", "type": "P", "read-only": true},

{"ifn": "org.allseenalliance.control.Mouse\*"}

]

},

{ "version": 5,

"peer": [{"type": "GUILD", "ID": "ParentsGuildGUID", "authority": "ownerPubKey"}],

"allow": [

{"obj": "/control/settings"}

]

}

]

}

#### Sample authorization data in a membership certificate for a provider

This sample shows a provider can emit a signal to members of the guild.

{

"provider": [

{ "version": 2,

"allow": [

{ "ifn": "org.allseenalliance.control.TV",

"mbr": "ChannelChanged", "type": "S"}

]

}

]

}

#### Sample authorization data for a consumer membership certificate

This sample shows the holder of the certificate has all the granted permissions provided in the provider policy. The guild ID comes from the membership certificate.

{

"consumer": [

{

"version": 32,

"allow": [

{"ifn": "\*"}

]

}

]

}

#### Sample authorization data for a consumer that restricts some access

This sample shows the holder of the membership certificate has all the granted permissions provided in the provider policy for the guild except a particular interface.

{

"consumer": [

{

"version": 1,

"allow": [

{"ifn": "\*"}

],

"deny": [{"ifn": "org.allseenalliance.control.Mouse\*"}]

}

]

}

#### Sample authorization data in a consumer policy file

This sample shows a consumer authorization data. In this policy, the consumer is not allowed to receive any signal with the exception of receiving a channel changed signal from any member of the guild LivingRoomGuild.

{

"consumer": [

{

"version": 11,

"peer": [{"type": "ANY"}],

"allow": [

{"ifn": "\*"}

],

"deny": [{"type": "S"}]

},

{

"version": 2,

"guild": [{"type": "GUILD", "ID": "LivingRoomGuildGUID", "authority": "guildPubKey"}],

"allow": [

{"ifn": "org.allseenalliance.control.TV",

"mbr": "ChannelChanged", "type": "S"}

]

}

]

}

#### Sample authorization data for a specific peer

This sample shows the holder of the access certificate can interact with the specific peer with the specific permissions.

{

"consumer": [

{"version": 1,

"peer": [{"type": "DSA", "ID":"D24174252B1DD243E0AAF903DE886ECA3FF38C973EAD6B81344F809200A3723900000000BB153866208721F15DC29FA2945266937096B2BAC1FB90EBDA943B2FFDAD2BDE0000000000000000"}],

"allow": [

{"ifn": "org.allseenalliance.control. ParentalControl"}

]

}

]

}

#### Sample authorization data for creating an admin

This sample shows how to create an admin.

{

"admin": [

{"version": 1,

"peer": [{"type": "DSA", "ID":"D24174252B1DD243E0AAF903DE886ECA3FF38C973EAD6B81344F809200A3723900000000BB153866208721F15DC29FA2945266937096B2BAC1FB90EBDA943B2FFDAD2BDE0000000000000000"}]

}

### Policy Templates (Work-In-Progress)

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

* Template name
* Description
* List of permission rules

#### Example of a Policy Template

This example shows an example of a policy template provided by a TV application:

{

"name": "Regular",

"desc": ["Regular access"],

"allow": [

{"ifn": "org.allseenalliance.control.OnOff"}

{"ifn": "org.allseenalliance.control.TV", "mbr": "Up"},

{"ifn": "org.allseenalliance.control.TV", "mbr": "Down"},

{"ifn": "org.allseenalliance.control.TV", "mbr": "Channel", "type": "P", "read-only": true},

{"ifn": "org.allseenalliance.control.Mouse\*"}

]

}

## Certificates

The following subsections detail the supported certificates.

### Policy certificate

Table 2‑2 lists the policy certificate fields. This type of certificate and its authorization data can be used as a policy certificate or a peer specific access certificate.

Table ‑. Policy certificate fields

| Field name | Description |
| --- | --- |
| version | * version is 1. * ECC curve is NIST P-256 * External Data digest algorithm is SHA-256. * DSA algorithm is ECC NIST P-256 DSA. |
| issuer | Issuer public key. |
| subject | Subject field holding the certificate holder’s public key. |
| validityFrom | Validity period. Subfield Valid From. It’s represented in seconds since EPOCH Jan 1, 1970. |
| validityTo | Validity period. Subfield ValidTo. It’s represented in seconds since EPOCH Jan 1, 1970. |
| delegate | Delegate flag. Must be set to false. |
| digest | Digest of the authorization data. |
| sig | DSA signature, which is computed over the fields from subject field to digest field by the issuer. |

### Membership certificate

Table 2‑3 lists the guild-specific certificate fields. This type of certificate and its authorization data can be used a membership certificate.

Table ‑. Guild-specific certificate fields

| Field name | Description |
| --- | --- |
| version | * version is 2. * ECC curve is NIST P-256 * External Data digest algorithm is SHA-256. * DSA algorithm is ECC NIST P-256 DSA. |
| issuer | Issuer public key. |
| subject | Subject field holding the certificate holder’s public key. |
| validityFrom | Validity period. Subfield Valid From. It’s represented in seconds since EPOCH Jan 1, 1970. |
| validityTo | Validity period. Subfield ValidTo. It’s represented in seconds since EPOCH Jan 1, 1970. |
| delegate | Delegate flag. |
| guild | Guild ID |
| digest | Digest of the authorization data. |
| sig | DSA signature, which is computed over the fields from subject field to digest field by the issuer. |

### User equivalence certificate

Table 2‑4 lists the user equivalence certificate fields. The subject will have all the privileges as the issuer.

Table ‑. User equivalence certificate fields

| Field name | Description |
| --- | --- |
| version | * version is 3. * ECC curve is NIST P-256 * External Data digest algorithm is SHA-256. * DSA algorithm is ECC NIST P-256 DSA. |
| issuer | Issuer public key. |
| subject | Subject field holding the certificate holder’s public key. |
| validityFrom | Validity period. Subfield Valid From. It’s represented in seconds since EPOCH Jan 1, 1970. |
| validityTo | Validity period. Subfield ValidTo. It’s represented in seconds since EPOCH Jan 1, 1970. |
| delegate | Delegate flag. Must be set to false. |
| sig | DSA signature, which is computed over the fields from subject field to delegate field by the issuer. |

### Identity certificate

Table 2‑5 lists the Identity certificate fields.

Table ‑. Identity certificate fields

| Field name | Description |
| --- | --- |
| version | * version is 4. * ECC curve is NIST P-256 * External Data digest algorithm is SHA-256. * DSA algorithm is ECC NIST P-256 DSA. |
| issuer | Issuer public key. |
| subject | Subject field holding the certificate holder’s public key. |
| validityFrom | Validity period. Subfield Valid From. It’s represented in seconds since EPOCH Jan 1, 1970. |
| validityTo | Validity period. Subfield ValidTo. It’s represented in seconds since EPOCH Jan 1, 1970. |
| delegate | Delegate flag. Must be set to false. |
| aliasLen | Length of the alias. The maximum length allowed is 40 bytes. |
| alias | Byte array for the alias. |
| digest | Digest of the identity data. |
| sig | DSA signature, which is computed over the fields from subject field to digest field by the issuer. |

#### Identity data format

The suggested Identity information can be expressed as vCard data using JSON format as described in [RFC 7095](http://tools.ietf.org/html/rfc7095).

### Guild equivalence certificate

Table 2‑6 lists the guild equivalence certificate fields.

Table ‑. Guild equivalence certificate fields

| Field name | Description |
| --- | --- |
| version | * version is 5. * ECC curve is NIST P-256 * External Data digest algorithm is SHA-256. * DSA algorithm is ECC NIST P-256 DSA. |
| issuer | Issuer public key. |
| subject | Subject field holding the certificate holder’s public key. |
| validityFrom | Validity period. Subfield Valid From. It’s represented in seconds since EPOCH Jan 1, 1970. |
| validityTo | Validity period. Subfield ValidTo. It’s represented in seconds since EPOCH Jan 1, 1970. |
| delegate | Delegate flag. Must be set to false. |
| guild | Locally definedguild ID. All membership certificates issued by the subject or its delegates will be treated as equivalences the locally definedguild. |
| sig | DSA signature, which is computed over the fields from subject field to guild field by the issuer. |

## 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

| Room | Devices | Notes |
| --- | --- | --- |
| Living room | TV, Set-top box, tablet, Network-attached Storage (NAS) | * All devices owned by Dad * All devices are accessible for the whole family * Tablet is managed by Dad, but the whole family can use it |
| Son’s bedroom | TV | * Owned and managed by son * Devices are allowed to interact with living room devices but the parent al control feature is denied. |
| Master bedroom | TV, tablet | * TV used by Mom and Dad only * Tablet used by Dad only * Devices can interact with living room devices |

### Users set up by Dad



Figure ‑. Use case - users set up by Dad

### Living room set up by Dad



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

### Son's bedroom set up by son



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

### Master bedroom set up by Dad



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

### Son can control different TVs in the house



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

### Living room tablet controls TVs in the house



Figure ‑. Use case - Living room tablet controls TVs

# 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.