AllJoyn™ Security 2.0 Feature

High-Level Design Document

Rev 1 Update 4

December 23, 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 a policy 13

2.3.5 Add an application to a guild 13

2.3.6 Add a user to a guild 14

2.3.7 Delegating membership certificate 15

2.3.8 Add a guild equivalence certificate to an application 17

2.3.9 Certificate revocation 17

2.3.10 Distribution of policy updates and membership certificates 18

2.3.11 Application Manifest 19

2.4 Access validation 21

2.4.1 Validation flow 21

2.4.2 Validating a consumer policy 22

2.4.3 Exchanging the membership certificates during session establishment 23

2.4.4 Anonymous session 24

2.4.5 Validating an admin user 26

2.4.6 Emitting a session-based signal 26

2.5 Authorization data format 27

2.5.1 The format is binary and exchanged between peers using AllJoyn marshalling 27

2.5.2 Format Structure 28

2.5.3 Policy Templates 32

2.6 Certificates 32

2.6.1 Main Certificate Structure 32

2.6.2 Identity certificate 33

2.6.3 Membership certificate 33

2.6.4 Guild equivalence certificate 34

2.7 Sample use cases 34

2.7.1 Users and devices 34

2.7.2 Users set up by Dad 35

2.7.3 Living room set up by Dad 36

2.7.4 Son's bedroom set up by son 37

2.7.5 Master bedroom set up by Dad 38

2.7.6 Son can control different TVs in the house 39

2.7.7 Living room tablet controls TVs in the house 40

3 Enhancements to Existing Framework 41

3.1 Crypto Agility Exchange 41

3.2 Permission NotifyConfig Announcement 42

4 Future Considerations 43

4.1 Broadcast signals and multipoint sessions 43

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 a policy 13

Figure 2‑6. Add an application to a guild 14

Figure 2‑7. Add a user to a guild 15

Figure 2‑8. Distribution of policy updates and certificate 19

Figure 2‑9: Building Policy using manifest 21

Figure 2‑10. Validation Flow 22

Figure 2‑11. Validating a consumer policy 23

Figure 2‑12. Exchange a trust profile 24

Figure 2‑13. Anonymous access 25

Figure 2‑14. Validating an admin user 26

Figure 2‑15. Validating a session-based signal 27

Figure 2‑16: Authorization Data Format Structure 28

Figure 2‑17. Use case - users set up by Dad 35

Figure 2‑18. Use case - living room set up by Dad 36

Figure 2‑19. Use case - son's bedroom set up by son 37

Figure 2‑20. Use case - master bedroom set up by Dad 38

Figure 2‑21. Use case – Son can control different TVs in the house 39

Figure 2‑22. Use case - Living room tablet controls TVs 40

Tables

Table 2‑1. Security 2.0 premises 8

# 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. |

## 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. The security Manger is optional because the permissions can be installed directly into the application.

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 2‑12. 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 GUID and 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.  A signed policy is a policy signed by an admin | * An admin can install, update, or remove a policy. * A newer signed policy can be installed by any peer. Developers can define policy templates to help the user with policy building. * Signed policy data must be encrypted if it is not delivered by the admin * 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. * An admin can query the existing policy installed in the application |
| 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 certificate * The membership authorization data is signed by the membership certificate issuer * 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. | * 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. |
| Security Manager |  | * Security Manager can push signed policy to subject application |

## 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. Claiming is first-come, first-claim action. That user becomes the admin. The procedure to make the device to become claimable again is manufacturer’s specific. There will be an API call that allow the device/application to make itself claimable again.

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



Figure 2‑2. 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 2‑3. 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.

A policy may contain a number of policy terms. Each term can be targeted to any user, a guild, or a particular user. Please refer to the section *Authorization data format* for more information.

### Install a policy

An admin can install a policy for the application.



Figure 2‑4. Install a 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 2‑6. 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 2‑7. 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 ‑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

### Certificate revocation

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

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

The application checks its installed policy for the Security Manager guild. If there is no such guild or the application can’t locate any of the Security Manager, 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. The relationship is defined by the matching issuer public key, subject public key, and guild ID.

### Distribution of policy updates and membership certificates

An admin uses the Security Manager to generate updated policy and membership certificates, encrypt the updated policy and deliver them to the Distribution Service for distribution to the applications he/she owns. These policy updates and certificates are signed by an admin, and the subject is the specific application. As the result, the application will trust the certificate and the signed policy updates.

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

Using the destination GUID, the Distribution Service discovers the target and attempts to install the updated policies and certificates.



Figure 2‑8. Distribution of policy updates and certificate

### Application Manifest

The main goal of a manifest is to inform the end-user which services an application will provide and consume. Manifest enforcement ensures the application cannot provide nor consume any unwarranted services. The trustworthy description of the interfaces shall be presented to the user in a human readable and localized fashion.

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

#### Manifest Format

The format of the manifest is similar to the format of the authorization data. Please refer to the section *Policy Templates* below for more information.

#### Trusted Description

The manifest data provided by the application does not contain any description. The description would be provided via HTTPS by a Service Description Server:

1. Provided by the developer using the reserve domain name of the interface name
2. Provided by the AllSeen Alliance

The developer must at least provide the description for the interface. An interface member listed in the manifest should have a description. If there is no description for the member, the interface description will be used in its place.

#### Manifest enforcement

As manifest are incorporated in the membership policy, no additional enforcement mechanism is required. The remote peer will intersect the rules in its local policy with the rules defined in the membership policy to enforce the application manifest.

#### Generating Policy and Membership Based on Manifest

The following flow shows how the Security Manager uses the manifest data provided by the application to generate local policy and membership policies.

#### 

Figure 2‑9: Building Policy using manifest

## Access validation

### Validation flow

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



Figure 2‑10. Validation Flow

### Validating a consumer policy

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



Figure 2‑11. Validating a consumer policy

### Exchanging the membership certificates during session establishment

During the AllJoyn session establishment, the peers exchange the membership certificates..



Figure 2‑12. 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 2‑13. Anonymous access

### Validating an admin user



Figure 2‑14. 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 2‑15. 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 signed buffer for DSA purpose.
3. The AllJoyn marshalling will be used to serialize the data for persistence purpose.
4. The parser will ignore any field that it does not support.

### Format Structure

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



Figure 2‑16: 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 |
| admins | Array of peer objects | no | The list of peers who have the admin privilege for the application. An admin peer becomes a certificate authority for the application. |
| terms | Array of Policy terms | no | List of policy terms. A term specifies the permissions the application can perform. |

Policy Term

| 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:   * ANY * GUID * GUILD | | keyInfo | structure of GUID and Public Key | no | The peer key info data. Depending on peer type, the keyInfo is:   * ANY – not applicable * GUID – the GUID and public key of the peer * GUILD – the GUID of the guild and the public key of the guild authority | |
| 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 \* 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. |

Interface Member Record

| Name | Data type | Required | List of values | Description |
| --- | --- | --- | --- | --- |
| mbr | string | no |  | Member name |
| type | number | 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. |
| action | byte | no |  | The action mask flag. The list of valid masks:   * 0x01: Denied * 0x02: Provide – allows to send signal, perform method calls and produce properties * 0x04: Observe – allows to receive signals and get properties * 0x08: Modify – Observe + Set properties + method calls |
| mutualAuth | boolean | no |  | Mutual authorization required. Both peers (local and remote) are required to be granted.  Default is yes. |



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

The following tables describe the rule enforcement.



| Message action | Required permission | | Affected member |
| --- | --- | --- | --- |
| Local peer | Remote peer |
| send GetProperty |  | Observe or  Modify allowed for consumer peer | property |
| receive GetProperty | Observe or  Modify allowed for consumer peer |  | property |
| send SetProperty |  | Modify allowed for consumer peer | property |
| receive SetProperty | Modify allowed for consumer peer |  | property |
| send method call |  | Modify allowed for consumer peer | method call |
| receive method call | Modify allowed for consumer peer |  | method call |
| send signal | Provide allowed |  | signal |
| receive signal | Provide allowed for emitter |  | 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.



























### Policy Templates

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:

* Specification version number
* List of permission rules



## 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.

### 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 an application with an identity alias.

The 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 identity data. For example, an identity VCard.

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 is part of a guild.

The guild identifier is encoded in the Organization Unit Name within the Subject Distinguished Name field.

The extensions include the following fields:

* CertificateType: the type of certificate within the AllSeen ecosystem. A membership certificate has certificate type equal to 2.
* AssociatedDigest: the digest of the associated authorization data.

Both the CertificateType and AssociatedDigest have custom OIDs 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) },

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 }

}











### Guild equivalence certificate

The guild equivalence certificate is used to map other certificates to a specific guild.

The guild identifier is encoded in the Organization Unit Name within the Subject Distinguished Name field.

The extensions include the following fields:

* CertificateType: the type of certificate within the AllSeen ecosystem. A guild equivalence certificate has certificate type equal to 3.

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) }

}



## 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 2‑17. Use case - users set up by Dad

### Living room set up by Dad



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

### Son's bedroom set up by son



Figure 2‑19. Use case - son's bedroom set up by son

### Master bedroom set up by Dad



Figure 2‑20. Use case - master bedroom set up by Dad

### Son can control different TVs in the house



Figure 2‑21. Use case – Son can control different TVs in the house

### Living room tablet controls TVs in the house



Figure 2‑22. 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 cypher 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 * SPKI based certificate | * Standard Client * Thin Client * Deprecated |
| ALLJOYN\_RSA\_KEYX | * AES\_128\_CCM\_8 * SHA256 * X.509 certificate | * Standard Client |
| ALLJOYN\_PIN\_KEYX | * AES\_128\_CCM\_8 | * Standard Client * 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 TSL 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 with 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 public key
3. The permission policy serial number

This signal is emitted when

1. The bus attachment is enable 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

# 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.