Weave Application Keys

Jay Logue
Google

Revision 5
2020/02/24

Introduction

This document describes Weave application keys, a mechanism for generating, disseminating and managing shared symmetric keys in a Weave network.

Goals

The goal of Weave application keys is to provide a common mechanism for generating, disseminating and managing a set of symmetric keys among groups of Weave-enabled nodes. These keys would be made available to applications for the purpose of authenticating peers, securing communications and storing encrypted data.

The Weave application keys mechanism is intended to serve as a reusable infrastructure component that facilitates the creation of robust and secure applications. As such its goals are to minimize redundant functionality and ensure a consistent level of security across all products and features.

A central feature of application keys is the ability to limit access to keys to a trusted set of nodes. In particular, group-shared application keys must only be accessible to nodes with a certain level of privilege, or that are expected to perform a certain role in the network. Barring software error or compromise of a privileged node, access to shared keys should be computationally infeasible for non-trusted parties.

Application keys must be shareable across all types and combinations of nodes, including Nest devices, end-user mobile devices running the Nest application and the Nest service itself. Importantly, for certain applications it must be possible to share keys across a set of nodes that *does not* include the Nest service.

Design Criteria

The following criteria influenced the design of the application keys mechanism:

- The mechanism should be simple, requiring modest implementation effort, while being flexible enough to support immediate and anticipated application requirements.
- The design should leverage existing architecture and communication paths to facilitate key distribution. In particular, the system should rely on WDM[1] protocols for the majority of key distribution requirements.
- The design should limit the amount of communication required to synchronize keys and key material, with the underlying goals of conserving device resources and minimizing complexity.
- The design should be implementable on highly constrained devices, including those with as little memory as 64KB RAM / 256KB Flash.

[1] Weave Data Management

In addition to the above, the design takes into consideration the following security criteria:

- Generated keys should be of sufficient strength for current and anticipated applications.
- Key rotation should be built-in and should apply to all applications in a consistent manner.
- The system should provide support for unilateral revocation of group membership.
- It should be possible to upgrade application security protocols without the need to distribute new keys, e.g. deriving protocol-specific keys from general purpose keys.

High-Level Design

The Weave application keys mechanism is a scheme for disseminating shared keys across a group of Weave nodes. These keys allow nodes to:

- Prove to each other that they are members of the associated group
- Exchange messages confidentially, and without fear of manipulation
- Encrypt data in such a way that it can only be decrypted by other members of the group
- Derive further cryptographic keys for use in other security settings (possibly involving non-Weave protocols and applications)

Application Groups

The application keys mechanism enforces group membership by ensuring that only legitimate members of the group have access to the associated keys. Key holders are organized into *application groups*, where each group corresponds to a logical set of abilities or privileges that members of the group are afforded. For example, a “Physical Access” group might represent those devices with the ability to lock and unlock doors in a structure, while a “Climate Control” group might represent all thermostats in the home. Groups can be made up of IoT devices and/or end-user mobile devices. Groups can also include the Nest service itself.

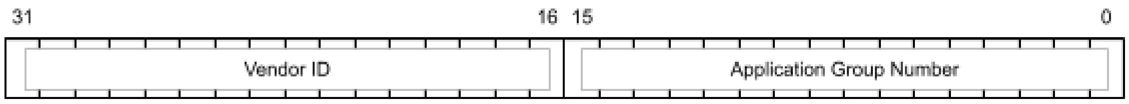
Membership in an application group is determined by a node’s intended role in the home network. For the most part this follows from the type of the node. But in some cases group membership is also influenced by the version of software running on a device.

Individual nodes can be members of multiple application groups simultaneously. The set of groups to which a device belongs can change over time, for example, as software updates introduce new functionality.

Up to 128 application groups can exist within a single home, and groups can be introduced or withdrawn over time as need arises.

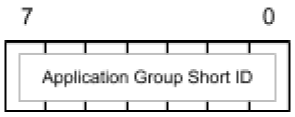
Application Group Ids

Application groups are identified by an *application group global id* (AGGID). Application group global ids are 32-bit integer values composed of a assigned vendor id and a vendor-specific group number. Vendors are expected to assign AGGIDs such that no two application groups have the same global id.



The AGGID value of 0 is reserved to represent a null or unspecified AGGID. (This corresponds to a Vendor ID of 0, which is also a reserved value).

Application groups can also be identified by an *application group short id* (AGSID) which is a 7 bit (0..127) integer that uniquely identifies an application group within the context of a single home. Application group short ids provide a compact representation of application groups that is suitable for use in key ids (see Key Ids below). AGSID are assigned by the Google Nest service, either automatically or by operator policy. Because of this, AGSIDs are not guaranteed to be the same across homes.



Key Generation

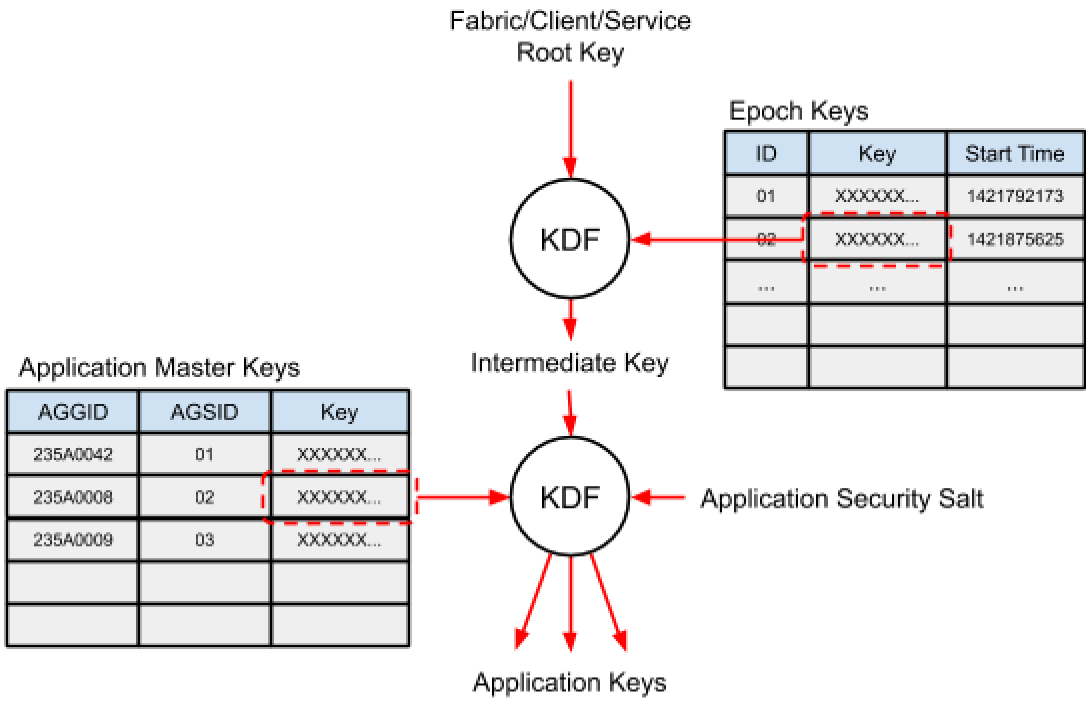
Membership in an application group confers access to the associated group's keys. These keys are produced by applying a sequence of key derivation algorithms to a set of input keys and salt values. Any number of keys can be generated from a common set of input values, allowing the mechanism to produce keys for a range of security algorithms and purposes.

For example, in some cases the key derivation algorithm may be used to generate Weave message encryption keys that, in turn, are used to secure the transmission of Weave messages between devices. In other cases data encryption or signing keys may be generated for application-specific use cases, such as the encryption of PIN codes for storage in the service. Ultimately the types and numbers of keys derived is a feature of the application group and the design of the application it serves.

In generic terms, the input keys to the generation process are referred to as *constituent keys*, while the derived output keys are referred to as *working keys*.

Group membership is enforced by limiting access to the constituent keys. Only nodes that possess all the constituent keys can derive the necessary works keys. Multiple levels of constituent keys are used to restrict access along different logical lines.

The following diagram shows the process by working keys are derived from the constituent key material:



Epoch Keys

ID	Key	Start Time
01	XXXXXX...	1421792173
02	XXXXXX...	1421875625
...	...	...

Application Master Keys

AGGID	AGSID	Key
235A0042	01	XXXXXX...
235A0008	02	XXXXXX...
235A0009	03	XXXXXX...

Input Key Material

At the highest level, a set of *root keys* serves to segregate application groups into three camps based on the types of nodes that are able to derive keys for the group.

A set of *application master keys* limits the key derivation process to nodes within the respective application groups. Access to application master keys is limited based on the functionality provided by a node and/or the privilege afforded to it. For example, certain home security devices, such as a security system or door lock, may have access to the “Physical Access” application master key, while devices such thermostats and smoke/CO sensors would probably not.

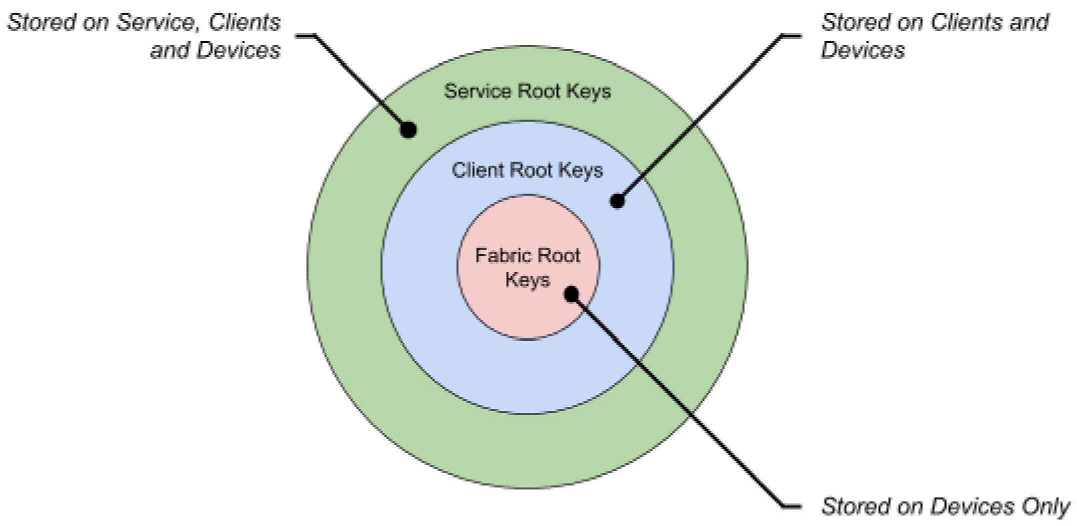
Key lifetime is limited by a set of *epoch keys* which govern the ability for members to derive application keys during particular periods of time. Epoch keys are rotated on a periodic basis, and denying access to updated version of these keys serves as a means to eject application group members by fiat.

Finally, a hard-coded *application scheme salt* is used to diversify the set of application keys according to the particular security scheme employed by the application. This salt value is hard-coded into the application’s implementation, and thus is distributed with the associated code. Should the application’s security mechanisms need to evolve (e.g. to upgrade encryption from AES-128 to AES-256), the security scheme salt can be changed to ensure that new keys will be derived for use in the new algorithm.

With the exception of the security scheme salt, all input key material is maintained on a per-home basis, aligning the scope of keys with the natural communication model for Weave devices.

Root Keys

There are three root keys per home, a *fabric root key*, a *client root key* and a *service root key*. All three keys are random 256-bit values generated uniquely for each home using a strong random number source. The choice of root key depends on the intended scope of the derived application key. The fabric root key is only available to Nest devices in the associated home, and therefore application keys derived from the fabric root key are only available to these devices. The client root key is available to both Nest devices within a particular home and to user client devices (mobiles, web browsers) that have access to that home. The service root key is available to Nest devices, user clients and to the service, allowing all parties to participate in key derivation.



The root key used to derive a particular application key is part of the definition of the associated application group. Each application group defines a desired scope for the use of its keys, which includes or excludes the service or client devices. The relationship between the application group and the choice of root key is established at group definition time, and is known *a priori* to all members of the group.

Fabric Root Keys

Fabric root keys are held solely by Nest devices within a fabric. Fabric root keys are accessible to client devices to facilitate pairing, but are never stored by them. Fabric root keys are *never* held by the service and do not flow through it.

The root key for a fabric is generated at fabric creation time, at the moment the first device forms the fabric. The fabric root key is generated randomly by the initial device and stored persistently in its memory. Thereafter, as new devices are added to the fabric, the fabric root key is passed from the existing device to the new device as part of the fabric provisioning process.

In cases where device joining is performed by a mobile device acting as a commissioner, transfer of the fabric root key is orchestrated by the commissioning client device. For automatic joining scenarios (e.g. Nest Thermostat to Heat Link), transfer occurs by direct device-to-device communication. In all cases, the fabric provisioning profile is used to affect the transfer.

Fabric root keys remain consistent throughout the lifetime of the fabric and are never rotated.

Client Root Keys

Client root keys are similar to fabric root keys with the exception that they can also be held by end-user client devices. Client root keys are not stored by the service but are accessible to it select operations such as exporting the key to a new client device.

Like fabric root keys, client root keys are generated by the Nest device that forms the fabric and are propagated to new devices by the pairing process. Client devices can receive client root keys by direct interaction with IoT devices in the home, or via the service, if the client device is out of the home.

Client root keys remain consistent throughout the lifetime of the fabric and are never rotated.

Service Root Keys

Service root keys are maintained by the service on a homebasis and are distributed to the devices and mobiles via transport. The service root key is created spontaneously by the service when the first device is paired into the home and stored persistently within service data store. Service root keys remain consistent throughout the lifetime of home and are never rotated.

Epoch Keys

Epoch keys provide a means for limiting the lifetime of derived application keys. They also provide a way for the service to revoke access to nodes that have been explicitly excluded from an application group (albeit after a period of time).

Epoch keys are generated by the service and stored on a per-home basis in the service data store. Each key is a random 256-bit value extracted from a strong random number source.

The key distribution service assigns a unique 3-bit epoch key id which identifies the key within the scope of the home. As new epoch keys are generated they are assigned new ids in monotonic order (modulo 8). In general, only 2 epoch keys are published by the key distribution service at any point in time–a *current epoch key* and a *new epoch key*.

Each epoch key has associated with it a start time that denotes the time at which nodes should begin using the key. Epoch key start times are expressed in Unix time with a precision of seconds.

Note that there is no end time associated with an epoch key.

Use of Epoch Keys

Nodes wishing to generate (send) secured data, or initiate secure sessions, must use application keys that are derived from the current epoch key (specifically, the epoch key with the *latest* start time that is not in the future).

At the same time, nodes accepting (receiving) and validating secured data, or responding to a request to initiate a secure session, must accept the use of *any* key derived from one of the then extant epoch keys. This requirement holds regardless of whether the start time for the key is in the future or the past. Based on this scheme, nodes continue to accept communication secured under an epoch key until that key is positively withdrawn by the key distribution service.

As a consequence of this policy it behooves applications to send a key id along with any secured data they communicate. This key id should positively identify the lineage of the key used to secure the data. This avoids the receiver having to try all possible valid epoch keys to determine the one from which the application key was derived. (See Key Ids for information on identifying keys).

Epoch Key Rotation

The key distribution service generates new epoch keys on a regular basis, giving each a unique id and adding them to the list of existing keys with a home. The start time for each new key is scheduled to occur after a configurable *key propagation interval*. The propagation internal is set sufficiently large such that all devices in the home can be expected to have synchronized with the new key list within that time.

As each new key is generated, the distribution service withdraws and destroys any previously generated epoch keys whose start time is earlier than the current time minus the propagation interval. This rotation scheme ensures that there are always at most two epoch keys published by the distribution service at any point in time.

The rotation rate for epoch keys is expected to be on the order of days to weeks for typical applications (however this rate should be configurable within the key distribution service to simplify testing).

Because of the relatively long rotation interval, and the overlap of active epoch keys, local clock drift within devices is generally not a concern.

Epoch Key Rotation without Time Synchronization

Although epoch keys are distributed with an associated start time, it is nonetheless possible for devices that do not maintain a synchronized clock to participate in key rotation. Specifically, upon receiving a new epoch key list from the key distribution service, such a device can note which of the two keys is the current epoch key by comparing their relative start times (the current epoch key having the earlier time). It can then use this key for all locally initiated security interactions until such time as it contacts the distribution service again.

This scheme requires the device to synchronize epoch keys with the key distribution service at a rate that is at least as fast as the configured key propagation interval.

Application Master Keys

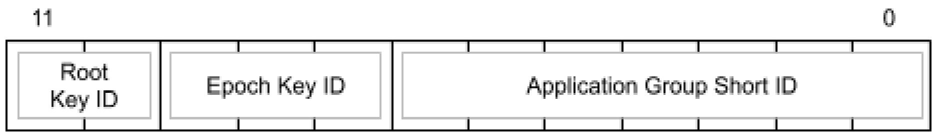
Application master keys govern membership in an application group. An application master key is a 256-bit random value generated and stored by the service on a per-home basis. Within each home a unique application master key exists for each application group that is active in that home. While the design allows for multitude of application groups to be active, in practice it is expected that the set of active application groups (and by extension the set of application master keys) will be uniform across all homes which contain the same types of devices.

Application master keys are generated either a device using them is paired into the home, or whenever a new application group is introduced, e.g. by means of a device firmware update. Once created, application master keys never change, and remain consistent until the application group is retired.

Key Ids

The application keys mechanism defines a compact numeric identifier for identifying application keys. An *application key id* is 12-bit value that identifies the key material used in the derivation of an application key. Given an application key id, and information about the application in which the keys are to be used, a node can derive the associated set of applications key values.

It consists of a 2-bit root key identifier, a 3-bit epoch key id and a 7-bit application group short id.



Application key ids are relative to the home in which the associated application key is used. Within the application key id, root keys are identified using the following values:

Root Key ID	Root Key Type
0	Fabric Root Key
1	Client Root Key

2	Service Root Key
3	*reserved*

The epoch key id field contains the numeric id value assigned to the epoch key by the service. Because epoch key rotate over time, the epoch key id field implicitly refers to the most recent epoch key to have been assigned.

Similar to the epoch key id field, the application group short id field contains the id value assigned by the service to the application group.

Application key ids are sized to be small enough to fix within the context of other key identifiers. In particular, application key ids are sized to fit within the Key Number field of a Weave message. When used in this context, the key id value allows a receiving node to derive the appropriate message encryption keys without additional information.

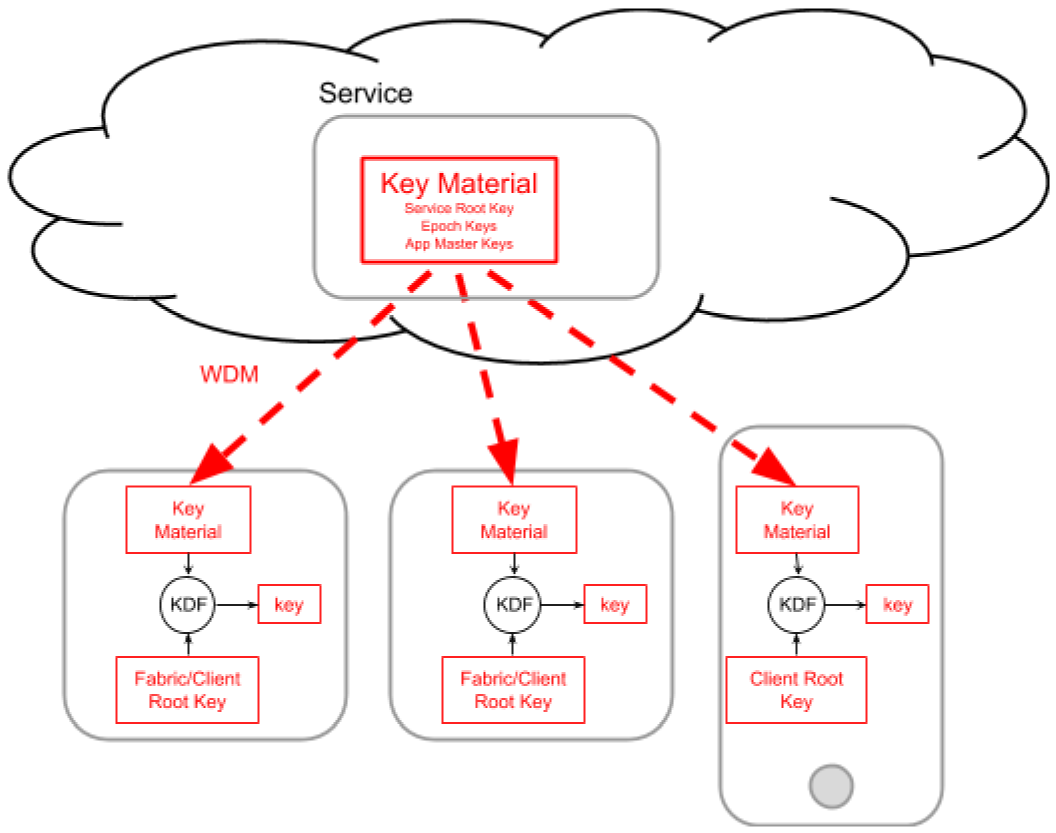
In other contexts the 12-bit key identifier might appear by itself, or in combination with information identifying the security protocol or algorithms.

Distribution of Key Material

The Weave Application Keys mechanism relies on a *key distribution service* to reliably distribute select constituent key material appropriate participants. Because the key distribution service itself is not in possession of all constituent keys, it is not necessarily a privileged participant in the associated application key groups. Nonetheless, the distribution service is responsible for enforcing access control such that only authorized nodes acquire the necessary keys.

With the exception of fabric root keys, key material is distributed to key holders using a data publishing mechanism such as WDM. In general, Key material is exposed via a single published object (trait in WDM) to which devices and mobiles subscribe and/or view. This object is referred to as the *key material* container. Since key material is scoped to the home, the key material container is indexed by the home's identifier.

The information exposed in the key material container includes the service root key, the epoch keys and the application master keys. Keys derived from these keys (the intermediate key and the application keys themselves) are never stored or published by the key distribution service.



Key Material Access Control

Access to application key material in the key material container is subject to standard authentication and authorization mechanisms. In particular, access to key material is restricted to authenticated users/devices that have privilege with respect to the associated home.

In addition to the above restriction, the key distribution service further restricts access to individual application master keys based on the group membership of the requestor. Only members of a particular application group are allowed to retrieve that group's application master key. The distribution service restricts access to master keys by blocking unauthorized keys from appearing in the key material container (as seen by the requesting node). Often this behavior will be based on the type of the requestor (e.g. a thermostat, a smoke/CO sensor, a mobile, etc.) and the version of software it runs. Together this information corresponds to the expected role of the node within the home, and therefore the privileges it needs to function.

In some cases, access to application master keys may be further contingent on explicit permission granted by the user. For example, a user may configure a node to perform a certain privileged operation, which in turn requires it to be a member of a particular application group. In this situation the key distribution service would include the application master key for the associated group in the response to this device based on information in the service data store representing the user's configuration.

Distribution of Fabric Root Keys to Devices

Fabric root keys are distributed among devices at pairing time, at the moment a new device joins a home. Transfer of the fabric root key happens as part of the fabric provisioning process, via the Weave Join Existing Fabric request. Once on a device, the fabric root key remains there for as long as the device is a member of the fabric (generally until the device is factory reset).

Distribution of Client Root Keys to Mobiles

Mobile devices acquire the client root key by retrieving it from a device in the fabric. For some mobiles this exchange occurs naturally as part of pairing a new device. However a mobile may also retrieve the client root key from a device at other times, e.g. when the user acquires a new phone and wishes to enable it to access their fabric. In either situation, the mobile extracts the client root key from a device that already exists in the fabric, using a new Weave request in the Weave Security Profile created expressly for this purpose.

To ensure the security of the client root key, access to the key by mobile devices can be subject to a multi-factor authentication process. Firstly, mobiles wishing to retrieve the client root key must authenticate to the device using a credential that confers authority with respect to the owner of the device. In one embodiment, this credential is the Weave Account Access Token. However other credential types may be supported.

Secondly, upon receiving a request for the client root key, the responding device will require that the request be received directly over its local WiFi interface, and not via one of its other network interfaces. Additionally, the request will not be allowed to be made over a network tunnel to the service. This serves as a kind of second factor, limiting access to the client root key to mobiles on the user's home's WiFi network. This also precludes the service from being co-opted into retrieving the client root key.

Once a mobile device has acquired a client root key it will cache the key for later use. Mobile devices must store cached client root keys in encrypted form, within a platform-specific secure storage mechanism. Preferably, copies of the client root key should be encrypted using a key derived from the user's account password. Cached client root keys may remain on the mobile device until the application is uninstalled, at which point all local copies should be actively destroyed.

Using Application Keys to Build Secure Applications

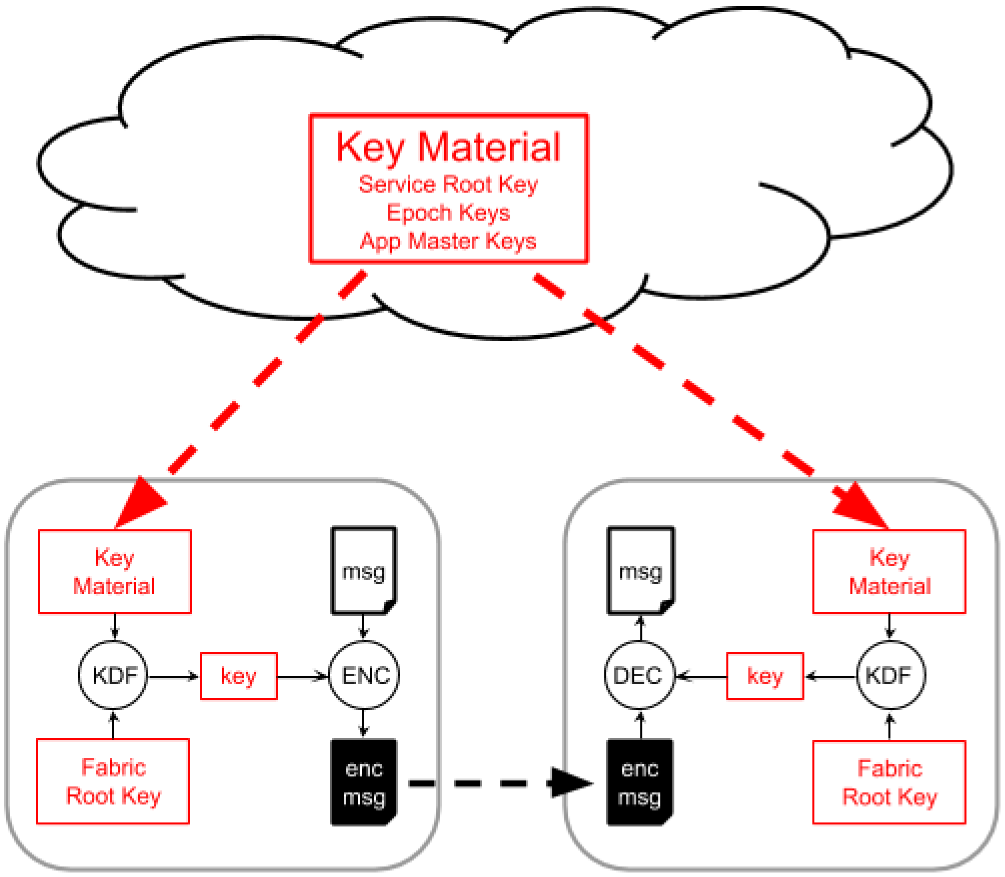
Application keys provides a common group shared-key scheme that can be used as the basis for security for various applications. In general, these applications will incorporate the application keys mechanism in one of three usage models:

- Securing direct node-to-node communication
- Encrypting shared data for storage in untrusted locations

- Establishing secure communication sessions

Direct Node-to-Node Communication

The application keys mechanism can be used to establish keys for direct node to node communication using Weave. In this situation, nodes derive Weave message encryption keys directly from the distributed key material, which they then use to exchange encrypted messages among their peer group members.



The direct node-to-node communication model has the benefit of not requiring explicit session establishment. Nodes can simply form and send messages at will to other group members without the need for a prior handshake. This usage model is limited, however, to nodes that are capable of maintaining global persistent message counter state. In general, this includes Weave-base IoT devices and mobiles running the Nest application. Because of the difficulty in maintaining highly synchronized state within a cloud-based system, the direct node-to-node communication model is generally unsuitable for communications involving device-to-cloud communication.

To maintain the security of the direct node-to-node communication model, key rotation must occur within the interval at which node message ids wrap. This time period ultimately depends on the rate at which individual nodes send messages. In practice, however, this rate is expected to be low, especially for sleepy devices. So this, coupled with the fact that message ids are 32-bits in size, means that message counter wrapping within the epoch key rotation period is very unlikely.

Examples of the direct node-to-node communication model include:

- Lock and unlock messages sent between a mobile and a door lock
- Sensor data updates between a sensor and a home security system
- Alarm messages between smoke/CO detectors
- Multicast time sync messages between devices in a home

Shared Key Session Establishment

For situations where direct node-to-node communication using application keys is not possible, application keys can be used in conjunction with a session establishment protocol to generate ephemeral session keys for use in securing Weave communication. Although the existing PASE protocol could be used for this purpose, the primary benefit would come from using a new session establishment protocol tailored to authentication using strong shared keys. Such a protocol would enable session establishment to take place with much less overhead as compared to CASE and PASE. Additionally, because communication occurs within the context of an ephemeral session, the protocol would be suitable for secure communication with the service.

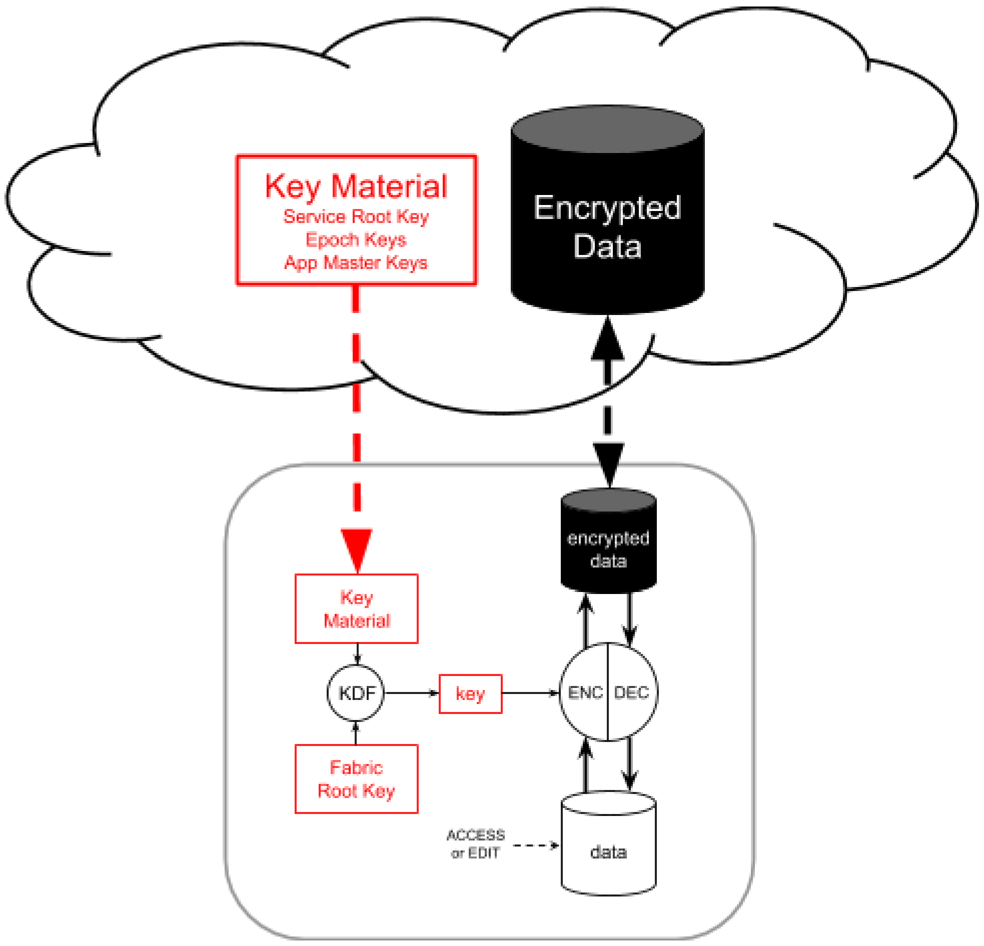
Shared Data Encryption

Applications can use the application keys mechanism to secure shared data that must be stored in an untrusted location. An example use case for this application would be the storage of WiFi or Thread network credentials in a cloud service.

To access the shared data, group members derive a general purpose data encryption key from the distributed group key material. They then use this key to locally decrypt and encrypt the shared data during access/update events. Key ids are stored along with the encrypted data so that group members can immediately tell which generation of key was used to secure the data.

To accommodate key rotation, group members arrange to re-encrypt shared data whenever a new generation of key becomes active. This action can be performed by a distinguished group member, or by “co-opetition” amongst members, with each one vying to re-encrypt the data first.

Depending on the type of root key selected, the service can be included or excluded in the data sharing model.



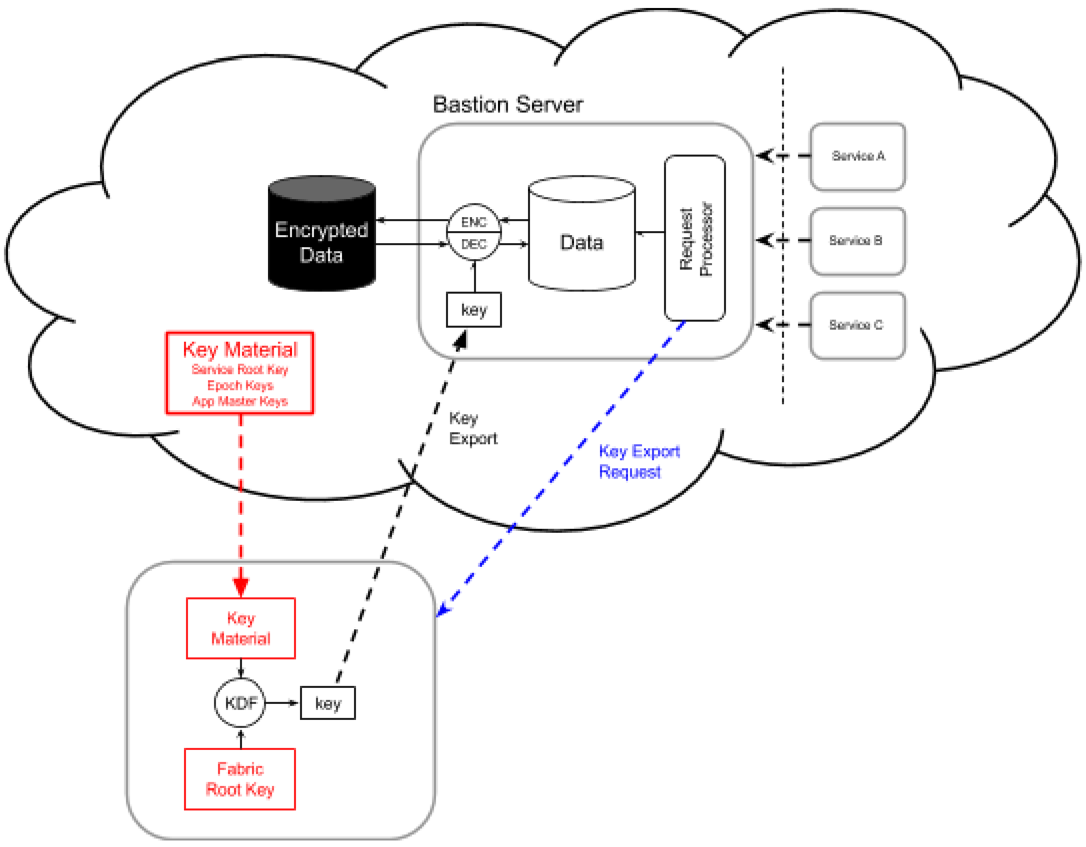
Shared Data Encryption with Limited Service Access

In some cases it is desirable to store encrypted data in the service in such a way that the service can access it, but only for a limited time and only with direct cooperation from a device in the home. An example of this use case would be the storage of user PIN data in the service. Typically PIN data would be accessed by mobiles, as well as certain classes of devices within the home (door locks, security systems). However the service may also offer users a PIN editing capability that is accessible via an external web interface.

In this model, the data encryption key is derived from the client root key, meaning that the service is not strictly a member of the associated application group. However, by means of a key export protocol, the service possesses the ability (with suitable authentication) to request the derived encryption key from a proper member node. Typically the node vending the encryption key will be a device in the home such as a security system.

Once in possession of the encryption key, the service uses it to access or update the secure data as necessary. Because the derived encryption key is subject to rotation, the exported key inherently has a finite lifetime, limiting the risk associated with its loss.

To further ensure the security of the application's data, the key distribution service also restricts access to the key, and the unencrypted form of the application data, to a specific *bastion server*. The bastion server is designed to perform a limited set of operations on the secure data at the behest of other service components. Preferably, the bastion server is situated with the larger service such that it has as little contact with other services as possible. While performing its task, the bastion server carefully limits the scope and lifetime of both the key and the decrypted data to ensure minimal risk of exposure. (Ideally, a single copy of the key and data are maintained in secure memory during the operation and never persisted to secondary storage). Once an operation is complete the bastion server thoroughly destroys all copies of the key and unencrypted data.



Revision History

1	2015/02/01	Initial revision.
2	2015/11/30	Added description of client root key. Revised identification scheme for application groups and master keys, introducing the notion of Application Group Global Ids, and Application Group Short Ids. Revised various diagrams for clarity and to reflect the above changes.
3	2016/04/15	Added note about AGGID value 0 being reserved as a “null” value.
4	2016/06/10	Cleaned up text that referred to mobile’s having access to fabric root keys.
5	2020/02/24	Updated content for general consumption. Added text covering topics that were raised in comments.