API Documentation

1. Introduction

The **Secret Service** API allows client applications to store secrets securely in a service running in the user's login session.

The secrets are usually stored in an encrypted manner by the service. The service may need to be unlocked by the user before the secrets become available for retrieval by client applications.

The Secret Service stores a secret along with a set of lookup attributes. The attributes can be used to look up and retrieve a secret at a later date. The lookup attributes are not treated as secret material, and the service may choose not to encrypt attributes when storing them to disk.

This API was designed by GNOME and KDE developers with the goal of having a common way to store secrets. Its predecessors are the desktop-specific APIs used by GNOME Keyring and KWallet.

2. Secrets

A secret is something an application wishes to store securely. A good example is a password that an application needs to save and use at a later date.

Within this API a secret value is treated as an **array of bytes**. It is recommended that a secret consist of user-readable text, although this API has no such requirement.

Applications wishing to store multiple values as part of a single secret, may choose to use a textual format to combine these values into one. For example, multiple values may be combined using the 'desktop' key file format, or XML.

Secrets may be encrypted when transferred to or from the client application.

The Secret structure encapsulates a secret value along with its transfer encryption parameters.

3. Collection and Items

Each secret is stored together with lookup attributes and a label. These together form an item. A group of items together form a collection. A collection is similar in concept to the terms 'keyring' or 'wallet'.

```
Collection
├── item = secret + lookup attributes + label
├── item = secret + lookup attributes + label
├── item = secret + lookup attributes + label
└── ...
```

Collections and items are represented as DBus objects, and **each has its own object path**. Under normal circumstances, the object path of a collection or item should not change for its lifetime. It is strongly recommended that client applications use lookup attributes to find items rather than recording the object path of a stored item. This allows maximum interoperability. An item or a collection may be initially in a locked state. When in a locked state the item or collection may not be modified in any way, and the secret may not be read. Client applications that require access to the secret of a locked item, or desire to modify a locked item, must unlock it before use. The service must prevent modification of locked collections or items. On such an invalid access the IsLocked error should be raised. Client applications without special requirements should store in the default collection. The default collection is always accessible through a specific object path. A new item can be created with the CreateItem() method on the Collection interface. When creating an item, the properties of the new item are specified. The service may ignore or change these properties when creating the item. When creating an item, the service may need to prompt the user for additional information. In this case, a prompt object is returned. It must be acted upon in order for the collection to be created. In this case, the result of the prompt will contain the object path of the new item. An item can be deleted by calling the Delete() method on the Item interface. When deleting an item, the service may need to prompt the user for additional information. In this case, a prompt object is returned. It must be acted upon in order for the item to be deleted. Client applications with special needs can create a new collection by calling the CreateCollection() method on the Service interface. When creating a collection, the properties of the new collection are specified. The service may ignore or change these properties when creating the collection. When creating a collection, the service may need to prompt the user for additional information. In this case, a prompt object is returned. It must be acted upon in order for the collection to be created. In this case, the result of the prompt will contain the object path of the new collection. A collection can be deleted by calling the Delete() method on the Collection interface. When deleting a collection, the service may need to prompt the user for additional information. In this case, a prompt object is returned. It must be acted upon in order for the collection to be deleted.

4. Aliases

Collections may be accessed via well known aliases. For example an alias called default tells applications which is the default collection to store secrets. The aliased collections will be available at a well-known DBus object path. If an application needs to create a collection with a given alias, this can be done in a race free fashion by specifying the alias parameter of the CreateCollection() method on the service interface. *If a collection with that alias already exists, then it will be returned instead of creating a new one*. For applications like password managers it can be useful to allow the user to configure which collection is associated with which well known alias. To alias or unalias a collection use the SetAlias() method on the service interface. Use the ReadAlias() method on the service interface to discover which collection is associated with a given alias.

5. Lookup Attributes

Attributes can and should be stored with a secret to facilitate lookup of the secret at a later date. An attribute consists of a **name**, and a **value**. Both parts are simple strings. The service may have additional requirements as to what can be present in an attribute name. It is recommended that attribute names are human readable, and kept simple for the sake of simplicity. During a lookup, attribute names and values are matched via **case-sensitive** string equality. It's important to remember that attributes are not part of the secret. Services implementing this API will probably store attributes in an unencrypted manner in order to support simple and efficient lookups. In order to search for items, use the SearchItems() method of

the Service interface. The matched items will be returned in two sets. The **unlocked** return value will contain the object paths of all the items that are not locked. The **locked** return value will contain object paths of items that are locked, which can be unlocked if desired.

The SearchItems() method of the Collection interface is similar, except for it only searches a single collection.

6. Sessions

A session is established between a client application and a service. A session is used to transfer secrets between the client application and the service. A session is established by calling the service's OpenSession() method. Once established, a session is bound to calling application's connection to the DBus session bus. A session is closed when the client application disconnects from the DBus session bus. Alternatively the client application can call the Close() method on the session interface. Once a session is closed all session specific negotiations will be dropped by the service. More than one session may opened by a client application, although this is not normally necessary.

7. Transfer of Secrets

Negotiation of Algorithms

Algorithm: plain

Algorithm: dh-ietf1024-sha256-aes128-cbc-pkcs7

To access or store secrets, use the GetSecret(), SetSecret() methods on the item interface, or the GetSecrets(), method on the service interface. **You must specify a session when retrieving or storing a secret**. *The session controls how the secret is encoded during transfer.* Since this is a D-Bus API, the data in all method calls and other accesses in this API will go through multiple processes, and may be cached arbitrarily by the OS or elsewhere. The Secrets API has provision to encrypt secrets while in transit between the service and the client application. The encryption is not envisioned to withstand man in the middle attacks, or other active attacks. It is envisioned to minimize storage of plain text secrets in memory and prevent storage plain text storage of secrets in a swap file or other caching mechanism. Many client applications may choose not to make use of the provisions to encrypt secrets in transit. In fact for applications unable to prevent their own memory from being paged to disk (eg: Java, C# or Python apps), transferring encrypted secrets would be an exercise of questionable value.

Negotiation of Algorithms

In order to encrypt secrets in transit, the service and the client application must agree on an algorithm, and some algorithm specific parameters (eg: a key). When the client application opens a session with the service, it calls the OpenSession() method on the service. The algorithms argument to the OpenSession() method specifies a set of algorithms to be used together for key agreement and encryption. The other arguments are algorithm specific. If a service does not support a specific set of algorithms, a **org.freedesktop.DBus.Error.NotSupported** error is returned, and the client is free to try another set of algorithms. The plain algorithm is almost always supported. An algorithm may require that the OpenSession() method is called multiple times in succession to be complete. Each iteration transfers algorithm specific data back forth between the service and the client. **The object path '/' is returned from OpenSession() when session negotiation is incomplete**. None of the algorithms documented in this initial version of the specification require multiple calls to OpenSession(). **When OpenSession() completes, it returns the session object path along with a valid session object path**. Once a session algorithm has been negotiated, it is used for all transfer a secrets whenever that session is specified along with the secret.

Algorithm: plain

- **Session algorithm**: plain
- Session input: empty string
- Session output: empty string
- Secret parameter: empty string

The plain algorithm does no encryption whatsoever. It is strongly recommended that a service implementing this API support the plain algorithm.

Algorithm: dh-ietf1024-sha256-aes128-cbc-pkcs7

- **Session algorithm**: dh-ietf1024-sha256-aes128-cbc-pkcs7
- Session input: Client DH pub key as an array of bytes
- Session output: Service DH pub key as an array of bytes
- Secret parameter: 16 byte AES initialization vector

DH key agreement rfc2631 is used to create a secret key using 1024 bit parameters of the standard IETF 'Second Oakley Group' rfc2409. The secret key is then digested into a 128-bit key appropriate for AES. This is done using HKDF rfc5869 with NULL salt and empty info, using the SHA-2 256 hash algorithm fips-180-3.2008. The secrets are encrypted using AES fips-197.2001 in cipher block chaining mode with pkcs7 style padding rfc2315. The public keys are transferred as an array of bytes representing an unsigned integer of arbitrary size, most-significant byte first (**big endian**) (e.g., the integer 32768 is represented as the 2-byte string 0x80 0x00).

8. Locking and Unlocking

Some items and/or collections may be marked as locked by the service. The secrets of locked items cannot be accessed. Additionally, locked items or collections cannot be modified by the client application. It's up to the service whether to unlock items individually, or collections as a whole. The client application should act as if it must unlock each item individually. A service may upon unlocking a collection, unlock all items in that collection. If a service is not able to unlock an item individually, it should treat a request to unlock an item as a request to unlock the collection that the item is in. A service may choose to unlock items or collections just for a single client application. Alternatively the service may choose to allow any client application to access items or collections unlocked by a another client application. A client application should always be ready to unlock the items for the secrets it needs, or objects it must modify. It must not assume that an item is already unlocked for whatever reason. A service may lock previously unlocked items for any reason at any time. Usually this is done in response to user actions, timeouts, or external actions (such as the computer sleeping). The inherent race conditions present due to this are unavoidable, and must be handled gracefully. In order to unlock an item or collection the service's Unlock() method is called with one or more DBus object paths of items or collections. The Unlock() will return the DBus object paths of objects it could immediately unlock without prompting. The Unlock() method may also return a prompt object. If a prompt object is returned, it must be acted upon in order to complete the unlocking of the remaining objects. The result of the prompt will contain the object paths that were successfully unlocked by the prompt. In order to lock an item or collection the service's Lock() method is called with one or more DBus object paths of

items or collections. The Lock() will return the DBus object paths of objects it could immediately lock without prompting. The Lock() method may also return a prompt object. If a prompt object is returned, it must be acted upon in order to complete the locking of the remaining objects. The result of the prompt will contain the object paths that were successfully locked by the prompt.

9. Prompts and Prompting

In order to complete various operations, such as unlocking a collection, the service may need to prompt the user for additional information, such as a master password. The prompts are displayed by the service on behalf of the client application. Operations that require a prompt to complete will return a prompt object. The client application must then call the Prompt() method of the prompt object to display the prompt. Client applications can use the window-id argument to display the prompt attached to their application window. Once the user provides the additional required information to the prompt, the service completes the operation that required the prompt. Then it emits the the Completed signal of the prompt object. The *result* argument of the signal contains operation an operation specific result. Either the user or the client application can dismiss a prompt. In this case the operation that required the additional information is cancelled. The client application can dismiss a prompt by calling the Dismiss() method of the prompt object. The Completed signal will be emitted with its dismissed argument set to TRUE. Once the Completed signal is emitted on a prompt object, it is no longer valid. Prompt objects are specific to the client application's connection to the DBus bus. Once an application disconnects, all its prompts are no longer valid. There is an inherent race between the Dismiss() method and the Completed signal. An application calling Dismiss() must be prepared to handle the fact that the Completed has already been emitted (although perhaps not received by the client). In addition the client must be prepared to handle the fact that the prompt object is no longer valid.

10. What's not included in the API

A service may implement additional DBus interfaces for further capabilities not included in this specification. Password management applications or other narrowly focused tools should make use of these when necessary. This specification does not mandate the use of master passwords to lock a collection of secrets. The service may choose to implement any method for locking secrets. This specification does not mandate any form of access control. The service may choose to allow certain applications to access a keyring, and others.

[TODO: complete]

11. Notes for Service Implementors

[TODO: complete]

Part II. D-Bus API Reference

12. Object Paths

The various DBus object paths used with the Secret Service API are designed to be human readable but not displayed to the user. The object path of an item or collection should not change for its lifetime, under normal circumstances.

`/org/freedesktop/secrets`

The object path for the service.

`/org/freedesktop/secrets/collection/xxxx`

The object path for a collection, where `xxxx` represents a possibly encoded or truncated version of the initial label of the collection.

`/org/freedesktop/secrets/collection/xxxx/iiii`

The object path for an item, where `xxxx` is the collection (above) and `iiii` is an auto-generated item specific identifier.

`/org/freedesktop/secrets/session/ssss`

The object path for a session, where `ssss` is an auto-generated session specific identifier.

`/org/freedesktop/secrets/aliases/default`

The default collection for client applications to store secrets is available under this object path in addition to its real object path (above). Other aliases may also be present.

13. Interfaces

- org.freedesktop.Secret.Service — The Secret Service manages all the sessions and collections.
- org.freedesktop.Secret.Collection — A collection of items containing secrets.
- org.freedesktop.Secret.Item — An item contains a secret, lookup attributes and has a label.
- org.freedesktop.Secret.Session — A session tracks state between the service and a client application.
- org.freedesktop.Secret.Prompt — A prompt necessary to complete an operation.

org.freedesktop.Secret.Service

org.freedesktop.Secret.Service — The Secret Service manages all the sessions and collections.

Synopsis

Methods

```
OpenSession (  IN String algorithm,
               IN Variant input,
               OUT Variant output,
               OUT ObjectPath result);

CreateCollection (  IN Dict<String,Variant> properties,
                    IN String alias,
                    OUT ObjectPath collection,
                    OUT ObjectPath prompt);


SearchItems (  IN Dict<String,String> attributes,
               OUT Array<ObjectPath> unlocked,
               OUT Array<ObjectPath> locked);

Unlock (  IN Array<ObjectPath> objects,
          OUT Array<ObjectPath> unlocked,
          OUT ObjectPath prompt);

Lock (  IN Array<ObjectPath> objects,
        OUT Array<ObjectPath> locked,
        OUT ObjectPath Prompt);

GetSecrets (  IN Array<ObjectPath> items,
              IN ObjectPath session,
              OUT Dict<ObjectPath,Secret> secrets);

ReadAlias (  IN String name,
             OUT ObjectPath collection);

SetAlias (  IN String name,
            IN ObjectPath collection);
```

Signals

```
CollectionCreated (OUT ObjectPath collection);
CollectionDeleted (OUT ObjectPath collection);
CollectionChanged (OUT ObjectPath collection);
```

Properties

```
READ Array<ObjectPath> Collections ;
```

Methods

org.freedesktop.Secret.Service.OpenSession

```
OpenSession (  IN String algorithm,
               IN Variant input,
               OUT Variant output,
               OUT ObjectPath result);
```

Open a unique session for the caller application.

- **algorithm**

  The algorithm the caller wishes to use.

- **input**

  Input arguments for the algorithm.

- **output**

  Output of the session algorithm negotiation.

- **result**

  The object path of the session, if session was created.

org.freedesktop.Secret.Service.CreateCollection

```
CreateCollection (  IN Dict<String,Variant> properties,
                    IN String alias,
                    OUT ObjectPath collection,
                    OUT ObjectPath prompt);
```

Create a new collection with the specified properties.

- **properties**

  Properties for the new collection. This allows setting the new collection's properties upon its creation. All READWRITE properties are useable. Specify the property names in full interface.Property form.

  Example 13.1. Example for properties

```
properties = { "org.freedesktop.Secret.Collection.Label": "MyCollection" }
```

- **alias**

  If creating this collection for a well known alias then a string like default. If a collection with this well-known alias already exists, then that collection will be returned instead of creating a new collection. Any readwrite properties provided to this function will be set on the collection.

  Set this to an empty string if the new collection should not be associated with a well known alias.

- **collection**

  The new collection object, or '/' if prompting is necessary.

- **prompt**

  A prompt object if prompting is necessary, or '/' if no prompt was needed.

org.freedesktop.Secret.Service.SearchItems

```
SearchItems (  IN Dict<String,String> attributes,
               OUT Array<ObjectPath> unlocked,
               OUT Array<ObjectPath> locked);
```

Find items in any collection.

- **attributes**

  Find secrets in any collection.

- **unlocked**

  Items found.

- **locked**

  Items found that require authentication.

org.freedesktop.Secret.Service.Unlock

```
Unlock (  IN Array<ObjectPath> objects,
          OUT Array<ObjectPath> unlocked,
          OUT ObjectPath prompt);
```

Unlock the specified objects.

- **objects**

  Objects to unlock.

- **unlocked**

  Objects that were unlocked without a prompt.

- **prompt**

A prompt object which can be used to unlock the remaining objects, or the special value '/' when no prompt is necessary.

org.freedesktop.Secret.Service.Lock

```
Lock (  IN Array<ObjectPath> objects,
  OUT Array<ObjectPath> locked,
  OUT ObjectPath Prompt);
```

Lock the items.

- **objects**

  Objects to lock.

- **locked**

  Objects that were locked without a prompt.

- **Prompt**

  A prompt to lock the objects, or the special value '/' when no prompt is necessary.

org.freedesktop.Secret.Service.GetSecrets

```
GetSecrets (  IN Array<ObjectPath> items,
              IN ObjectPath session,
              OUT Dict<ObjectPath,Secret> secrets);
```

Retrieve multiple secrets from different items.

- **items**

  Items to get secrets for.

- **session**

  The session to use to encode the secrets.

- **secrets**

  Secrets for the items.

org.freedesktop.Secret.Service.ReadAlias

```
ReadAlias (  IN String name,
             OUT ObjectPath collection)
```

Get the collection with the given alias.

- **name**

  An alias, such as 'default'.

- **collection**

  The collection or the the path '/' if no such collection exists.

org.freedesktop.Secret.Service.SetAlias

```
SetAlias (  IN String name,
            IN ObjectPath collection);
```

Setup a collection alias.

- **name**

  An alias, such as 'default'.

- **collection**

  The collection to make the alias point to. To remove an alias use the special value '/'.

Signals

org.freedesktop.Secret.Service.CollectionCreated

```
CollectionCreated (OUT ObjectPath collection);
```

A collection was created.

- **collection**

  Collection that was created

org.freedesktop.Secret.Service.CollectionDeleted

```
CollectionDeleted (OUT ObjectPath collection);
```

A collection was deleted.

- **collection**

  Collection that was deleted

org.freedesktop.Secret.Service.CollectionChanged

```
CollectionChanged (OUT ObjectPath collection);
```

A collection was changed.

- **collection**

  Collection that was changed

D-Bus Properties

Accessed using the org.freedesktop.DBus.Properties interface.

```
READ Array<ObjectPath> Collections ;
```

The object paths of all collections (ie: keyrings)

org.freedesktop.Secret.Collection

org.freedesktop.Secret.Collection — A collection of items containing secrets.

Synopsis

Methods

```
Delete (  OUT ObjectPath prompt);

SearchItems (  IN Dict<String,String> attributes,
               OUT Array<ObjectPath> results);

CreateItem (  IN Dict<String,Variant> properties,
              IN Secret secret,
              IN Boolean replace,
              OUT ObjectPath item,
              OUT ObjectPath prompt);
```

Signals

```
ItemCreated (OUT ObjectPath item);
ItemDeleted (OUT ObjectPath item);
ItemChanged (OUT ObjectPath item);
```

Properties

```
READ Array<ObjectPath> Items ;
READWRITE String Label ;
READ Boolean Locked ;
READ UInt64 Created ;
READ UInt64 Modified ;
```

Methods

org.freedesktop.Secret.Collection.Delete

```
Delete (OUT ObjectPath prompt);
```

Delete this collection.

- **prompt**

  A prompt to delete the collection, or the special value '/' when no prompt is necessary.

org.freedesktop.Secret.Collection.SearchItems

```
SearchItems (  IN Dict<String,String> attributes,
               OUT Array<ObjectPath> results);
```

Search for items in this collection matching the lookup attributes.

- **attributes**

  Attributes to match.

- **results**

  Items that matched the attributes.

org.freedesktop.Secret.Collection.CreateItem

```
CreateItem (  IN Dict<String,Variant> properties,
              IN Secret secret,
              IN Boolean replace,
              OUT ObjectPath item,
              OUT ObjectPath prompt);
```

Create an item with the given attributes, secret and label. If replace is set, then it replaces an item already present with the same values for the attributes.

- **properties**

  The properties for the new item.

  Properties for the new item. This allows setting the new item's properties upon its creation. All READWRITE properties are useable. Specify the property names in full interface.Property form.

  Example 13.2. Example for properties

```
  properties = {
               "org.freedesktop.Secret.Item.Label": "MyItem",
               "org.freedesktop.Secret.Item.Attributes": {
                           "Attribute1": "Value1",
                           "Attribute2": "Value2"
                     }
            }
```

  Note: Please note that there is a distinction between the terms Property, which refers to a D-Bus properties of an object, and Attribute, which refers to one of a secret item's string-valued attributes.

- **secret**

  The secret to store in the item, encoded with the included session.

- **replace**

  Whether to replace an item with the same attributes or not.

- **item**

  The item created, or the special value '/' if a prompt is necessary.

- **prompt**

  A prompt object, or the special value '/' if no prompt is necessary.

Signals

org.freedesktop.Secret.Collection.ItemCreated

```
ItemCreated (OUT ObjectPath item);
```

A new item in this collection was created.

- **item**

  The item that was created.

org.freedesktop.Secret.Collection.ItemDeleted

```
ItemDeleted (OUT ObjectPath item);
```

An item in this collection was deleted.

- **item**

  The item that was deleted.

org.freedesktop.Secret.Collection.ItemChanged

```
ItemChanged (OUT ObjectPath item);
```

An item in this collection changed.

- **item**

  The item that was changed.

D-Bus Properties

Accessed using the org.freedesktop.DBus.Properties interface.

```
READ Array<ObjectPath> Items ;
```

Items in this collection.

```
READWRITE String Label ;
```

The displayable label of this collection.

```
READ Boolean Locked ;
```

Whether the collection is locked and must be authenticated by the client application.

```
READ UInt64 Created ;
```

The unix time when the collection was created.

```
READ UInt64 Modified ;
```

The unix time when the collection was last modified.

org.freedesktop.Secret.Item

org.freedesktop.Secret.Item — An item contains a secret, lookup attributes and has a label.

Synopsis

Methods

```
Delete (  OUT ObjectPath Prompt);

GetSecret (  IN ObjectPath session,
             OUT Secret secret);

SetSecret (  IN Secret secret);
```

Properties

```
READ Boolean Locked ;
READWRITE Dict<String,String> Attributes ;
READWRITE String Label ;
READ UInt64 Created ;
READ UInt64 Modified ;
```

Methods

org.freedesktop.Secret.Item.Delete

```
Delete (OUT ObjectPath Prompt);
```

Delete this item.

- **Prompt**

  A prompt object, or the special value '/' if no prompt is necessary.

org.freedesktop.Secret.Item.GetSecret

```
GetSecret (  IN ObjectPath session,
             OUT Secret secret);
```

Retrieve the secret for this item.

- **session**

  The session to use to encode the secret.

- **secret**

  The secret retrieved.

org.freedesktop.Secret.Item.SetSecret

```
SetSecret (IN Secret secret);
```

Set the secret for this item.

- **secret**

  The secret to set, encoded for the included session.

D-Bus Properties

Accessed using the org.freedesktop.DBus.Properties interface.

```
READ Boolean Locked ;
```

Whether the item is locked and requires authentication, or not.

```
READWRITE Dict<String,String> Attributes ;
```

The lookup attributes for this item.

```
READWRITE String Label ;
```

The displayable label for this item.

```
READ UInt64 Created ;
```

The unix time when the item was created.

```
READ UInt64 Modified ;
```

The unix time when the item was last modified.

org.freedesktop.Secret.Session

org.freedesktop.Secret.Session — A session tracks state between the service and a client application.

Synopsis

Methods

```
Close (void);
```

Methods

org.freedesktop.Secret.Session.Close

```
Close (void);
```

Close this session.

org.freedesktop.Secret.Prompt

org.freedesktop.Secret.Prompt — A prompt necessary to complete an operation.

Synopsis

Methods

```
Prompt (IN String window-id);
Dismiss (void);
```

Signals

```
Completed (  OUT Boolean dismissed,
             OUT Variant result);
```

Methods

org.freedesktop.Secret.Prompt.Prompt

```
Prompt (IN String window-id);
```

Perform the prompt.

- **window-id**

  Platform specific window handle to use for showing the prompt.

org.freedesktop.Secret.Prompt.Dismiss

```
Dismiss (void);
```

Dismiss the prompt.

Signals

org.freedesktop.Secret.Prompt.Completed

```
Completed (  OUT Boolean dismissed,
             OUT Variant result);
```

The prompt and operation completed.

- **dismissed**

  Whether the prompt and operation were dismissed or not.

- **result**

  The possibly empty, operation specific, result.

14. Types

Struct types

Secret

The Secret type holds a (possibly encoded) secret.

Arrays of Secret don't generally make sense.

```
struct Secret {
  ObjectPath session ;
  Array<Byte> parameters ;
  Array<Byte> value ;
  String content_type ;
}
```

- **session**

  The session that was used to encode the secret.

- **parameters**

  Algorithm dependent parameters for secret value encoding.

- **value**

  Possibly encoded secret value

- **content_type**

  The content type of the secret. For example: 'text/plain; charset=utf8'

Map types

ObjectPath_Secret_Map

A mapping from object-paths to Secret structs

- **Members**
  - **Key**: D-Bus object-path
  - **Value**: A secret

15. Errors

Errors

Errors returned by the Secret Service API.

org.freedesktop.Secret.Error.IsLocked

The object must be unlocked before this action can be carried out.

org.freedesktop.Secret.Error.NoSession

The session does not exist.

org.freedesktop.Secret.Error.NoSuchObject

No such item or collection exists.

org.freedesktop.DBus.Error.NotSupported

Service does not support a specific set of algorithms for encryption.

References

- [rfc2315] IETF RFC 2315: PKCS #7: Cryptographic Message Syntax Version 1.5
- [rfc2409] IETF RFC 2409: The Internet Key Exchange (IKE)
- [rfc2631] IETF RFC 2631: Diffie-Hellman Key Agreement Method
- [rfc5869] IETF RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
- [fips-180-3.2008] NIST FIPS PUB 180-3: Secure Hash Standard (SHS), October 2008
- [fips-197.2001] NIST FIPS PUB 197: Advanced Encryption Standard (AES), November 2001