Purebred Key Sharing 2025

Purebred is the derived credential issuing system for the United States (U.S.) Department of Defense (DoD). Since 2016, the Purebred app for iOS has featured a custom "key sharing" interface that allows PKCS #12 objects and the corresponding passwords to be shared from the Purebred app to unrelated apps via the iOS file provider extension APIs and the system pasteboard. Two sample apps were prepared to enable application developers to test integration with the key sharing interface and to demonstrate usage: SampleKeyProvider and KeyShareConsumer. Since 2020, the Purebred app for iOS has featured a persistent token extension that enables other apps to use keys provisioned via Purebred without exporting and sharing the private keys. The persistent token interface is the preferred way to exercise Purebred-provisioned keys on iOS devices. As with key sharing, two sample apps were prepared to enable application developers to test integration with the persistent token interface and to demonstrate usage: CtkProvider and CtkConsumer.

In the years since 2016, several APIs that underpin the key sharing mechanism have been deprecated. This document describes an updated key sharing implementation that does not rely on any APIs that are deprecated at this time. Unfortunately, these changes are not backwards compatible and result in changes to the user experience. Two new sample apps, SampleKeyProvider2025 and KeyShareConsumer2025, have been published to facilitate testing relative to the updated key sharing interface.

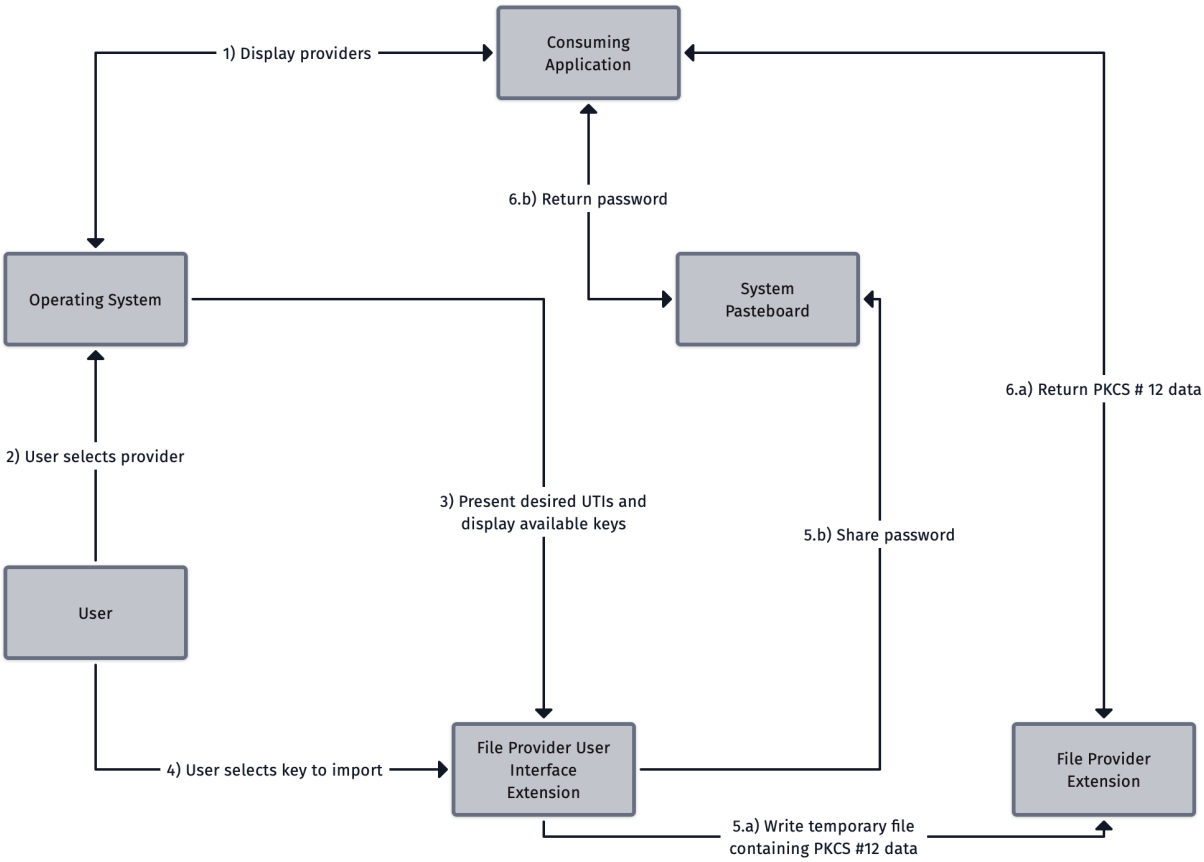
Throughout this document, the key sharing mechanism deployed in 2016 is referred to as "legacy key sharing" and the key sharing mechanism that will be deployed in 2025 is referred to as "key sharing 2025."

Legacy Key Sharing

In the legacy key sharing implementation, six actors play active roles:

- Consuming application
- Operating system
- File provider extension
- File provider user interface extension
- System pasteboard
- User

The host application is not listed here. Though it contains the two extensions and handles the provisioning of derived credentials that are the target of key sharing, it is not an active participant in key sharing. However, the host application, file provider extension, and file provider user interface extension must all share a key chain access group. The key chain is not shown here, but is the source for keys returned from the file provider and, in most cases, is the destination for keys within the consuming application (albeit using a different key chain access group). The diagram below shows the primary activities associated with legacy key sharing.



Legacy Key Sharing Architecture

1) Display providers

Legacy key sharing is initiated when a consuming application launches an instance of UIDocumentPickerViewController that was initialized with a list of desired uniform type identifiers (UTIs). Note, the `init(documentTypes:in:)` method has been deprecated.

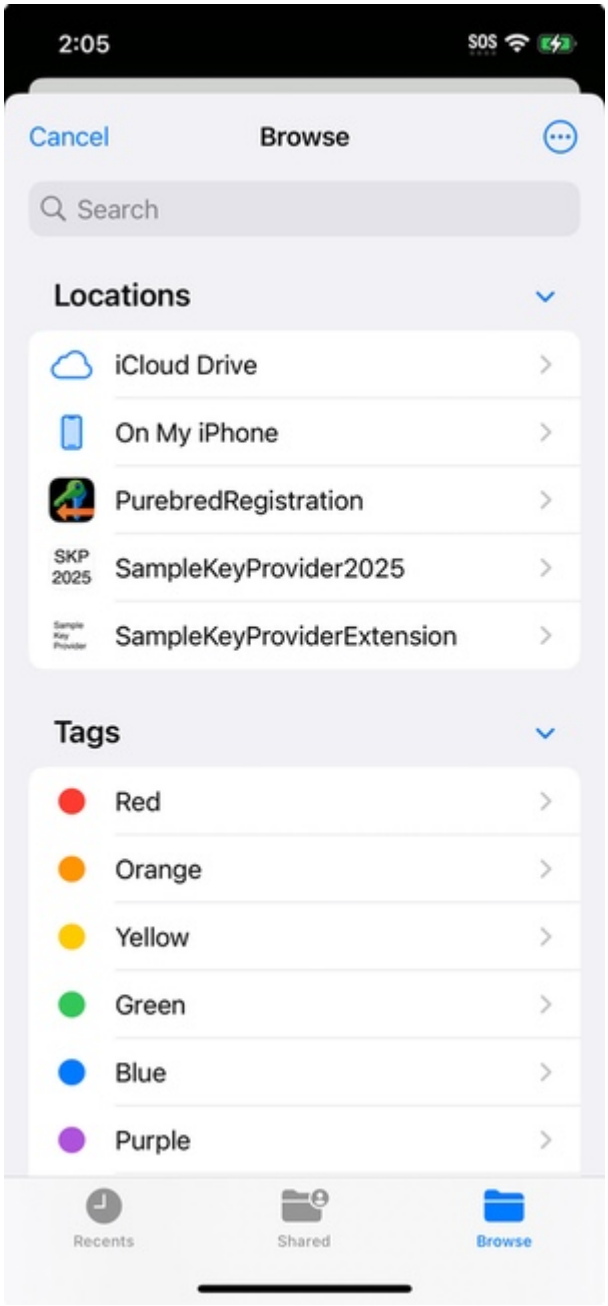
```
UIDocumentPickerViewController *documentPicker =
[[UIDocumentPickerViewController alloc] initWithDocumentTypes:utis
inMode:UIDocumentPickerModeOpen];
documentPicker.delegate = self;
documentPicker.modalPresentationStyle = UIModalPresentationFormSheet;
[self presentViewController:documentPicker animated:YES completion:nil];
```

This causes the operating system to display a view that lists sources from which keys may be imported or a previously used source, if the mechanism has been used previously.

In the Key Share Consumer app, this occurs in the click handler for the `Import Key` button.

2) User selects provider

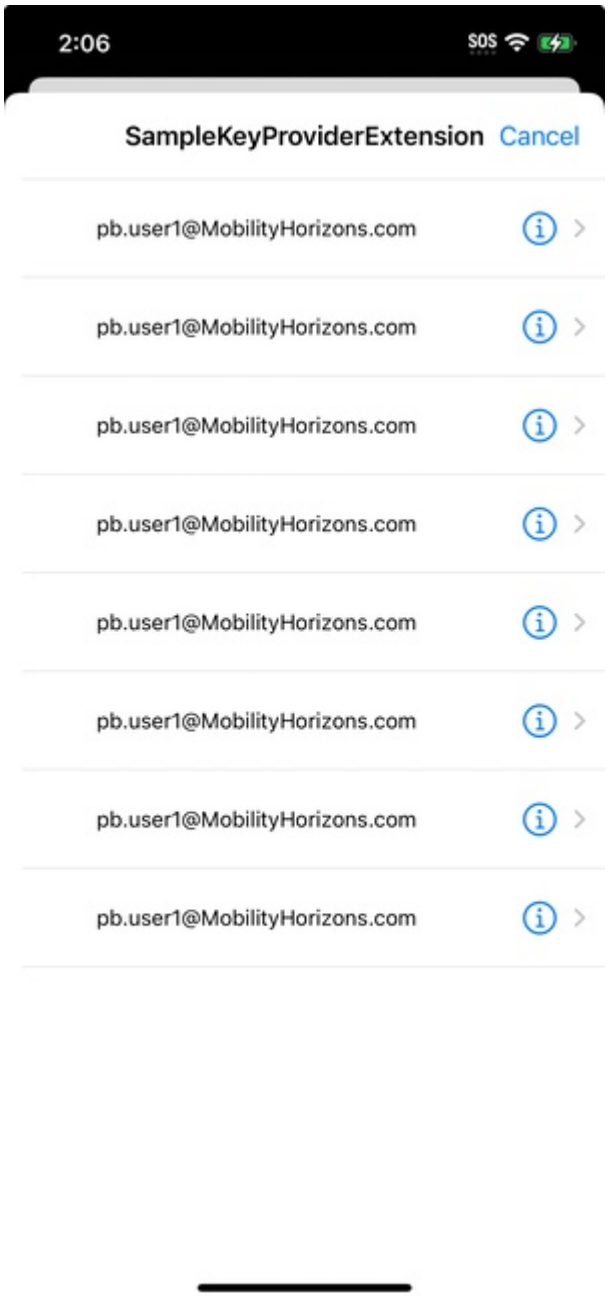
The user chooses a source, which may initially be implicit as a previously used provider is displayed by default. The screenshot below shows an example of a list of providers presented for user selection.



Provider Selection in Legacy Key Sharing

3) Present desired UTIs and display available keys

After the user makes a selection, the operating system invokes the subclass of UIDocumentPickerExtensionViewController exposed by the file provider user interface extension that is included in the Purebred or Sample Key Provider app. The extension uses the list of UTIs to prepare a tailored list of items that the user may import. The screen shot below shows an example of a list of keys displayed by a provider.

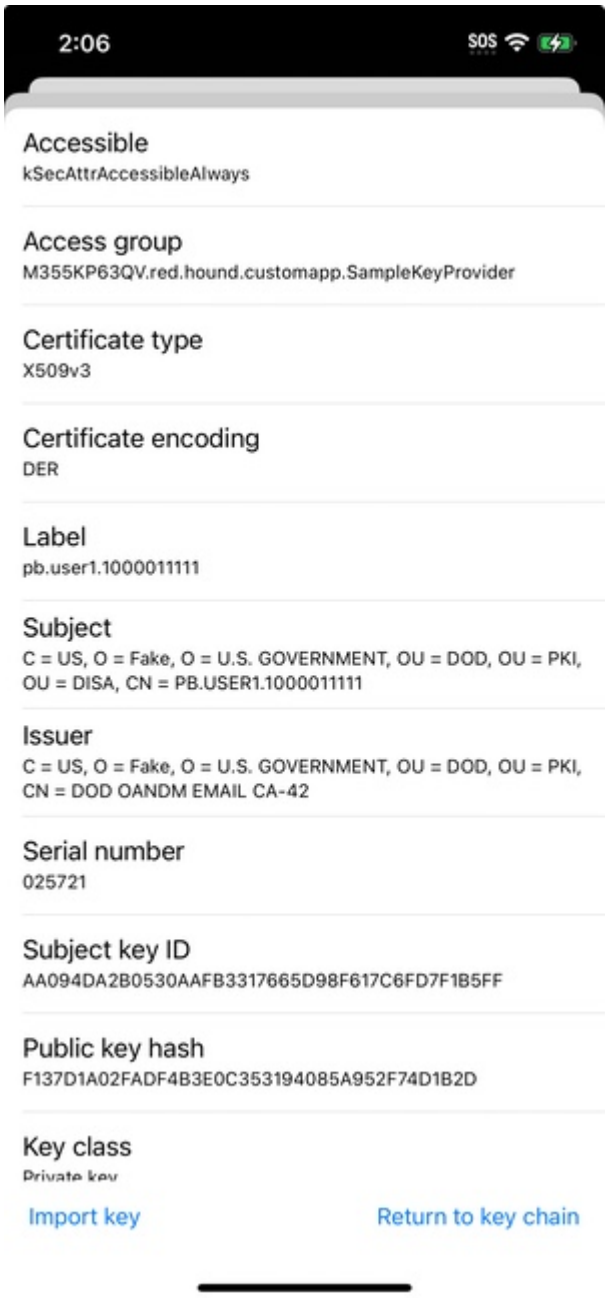


Key Display in Legacy Key Sharing

In the Sample Key Provider app, this occurs in the prepareForPresentationInMode method using the validTypes variable inherited from `UIDocumentPickerExtensionViewController` to limit the keys that are displayed. Note, the `validTypes` variable has been deprecated.

4) User selects keys

The user clicks on a row in the resulting table to display key details, as shown below, then may choose to import a key or return to the list to choose a different key or abandon the import operation.



Key Details in Legacy Key Sharing

5.a) Write temporary file containing PKCS #12 data

After the user selects a key, the file provider user interface extension exports the selected key from the key chain, prepares a PKCS #12 object using a randomly generated password, and writes the result to a temporary file. After the file is staged, dismissGrantingAccess is called, which results in the file provider extension being invoked. Note, `dismissGrantingAccess` has been deprecated.

In the Sample Key Provider app, this occurs in the import method that is invoked when the user makes a selection.

5.b) Share password

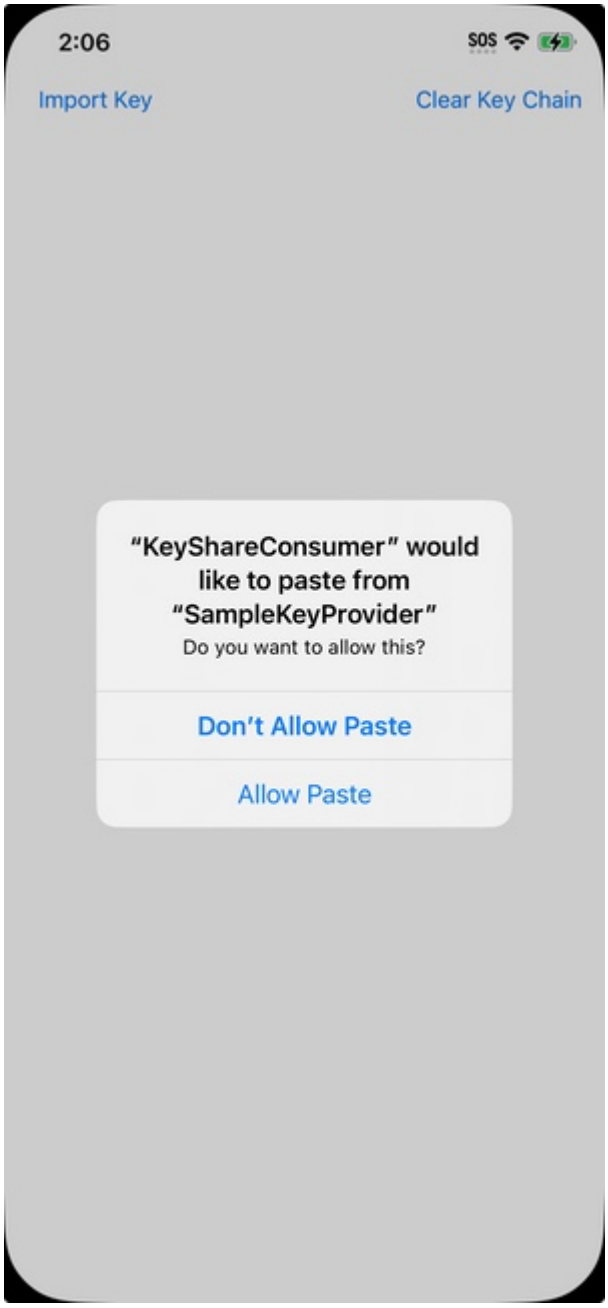
As part of the PKCS #12 preparation, the randomly generated password is written to the system pasteboard for retrieval by the consuming application. It is worth noting that access to the system pasteboard is permitted from file provider user interface extensions but not from file provider extensions.

6.a) Return PKCS #12 data

The file provider extension uses an NSFileCoordinator instance to facilitate providing the consumming application access to the PKCS #12 file created by the file provider user interface extension.

6.b) Return password

The consuming application reads the password from the system pasteboard. The user must authorize the app to access the password, as shown in the screenshot below. The application can use this password to process the PKCS #12 data that was obtained.



Allow Pasteboard Access

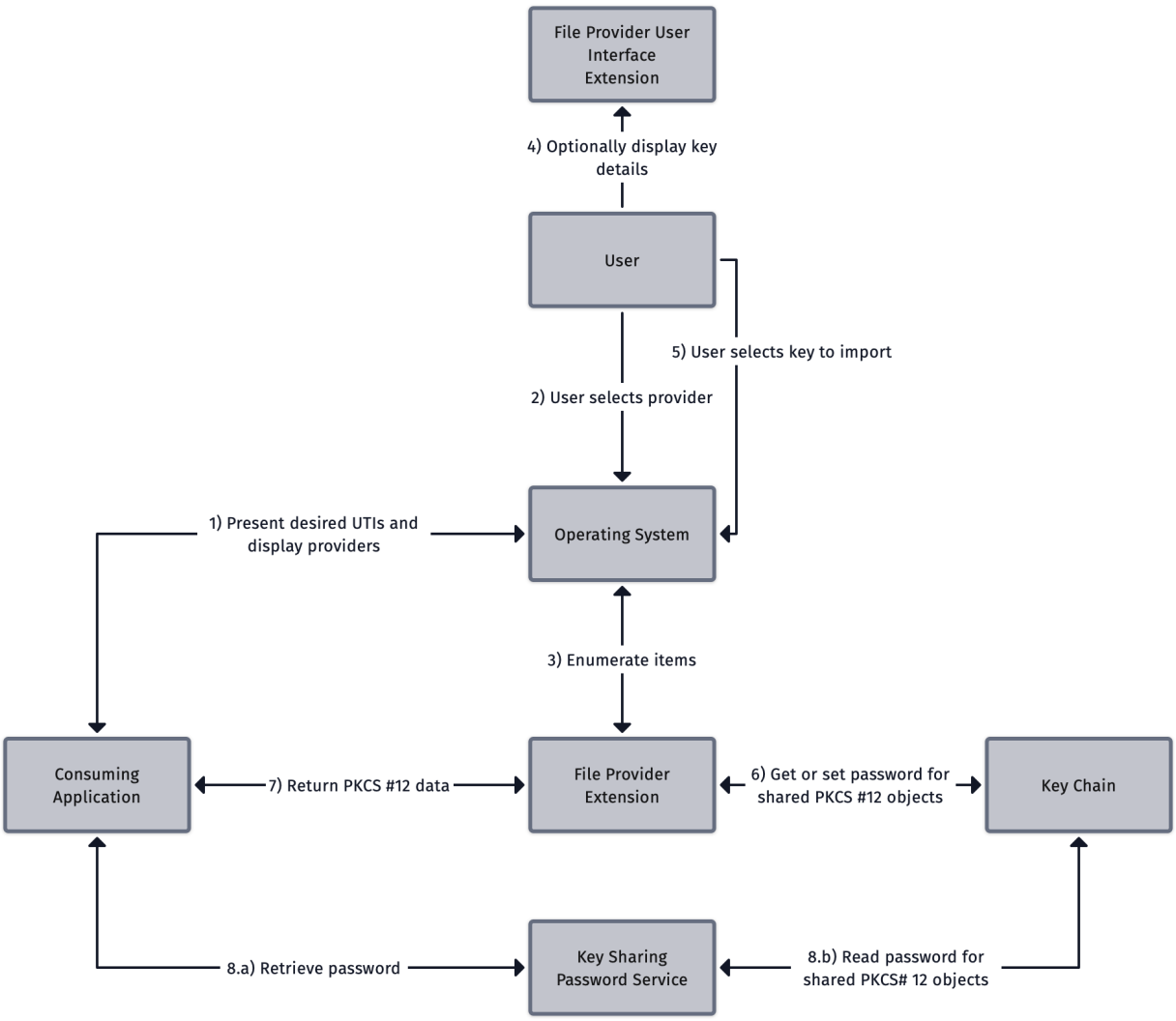
In the Key Share Consumer app, this occurs in the didPickDocumentsAtURLs implementation.

Key Sharing 2025

In the key sharing 2025 implementation, seven actors play active roles:

- Consuming application
- Operating system
- File provider extension
- File provider user interface extension
- Key sharing password service
- Key chain
- User

The host application is not listed here. Though it contains the two extensions and handles the provisioning of derived credentials that are the target of key sharing, it is not an active participant in key sharing. Unlike legacy key sharing, the key chain is listed here as a store for the password used to encrypt PKCS #12 objects. As with legacy key sharing, the key chain is also the source for keys returned from the file provider and, in most cases, is the destination for keys within the consuming application. The host application, file provider extension, and file provider user interface extension must all share a key chain access group. The diagram below shows the primary activities.



Key Sharing 2025 Architecture

1) Present desired UTIs and display providers

Similar to legacy key sharing, key sharing 2025 is initiated when a consuming application launches an instance of UIDocumentPickerViewController, passing a list of desired uniform type identifiers (UTIs).

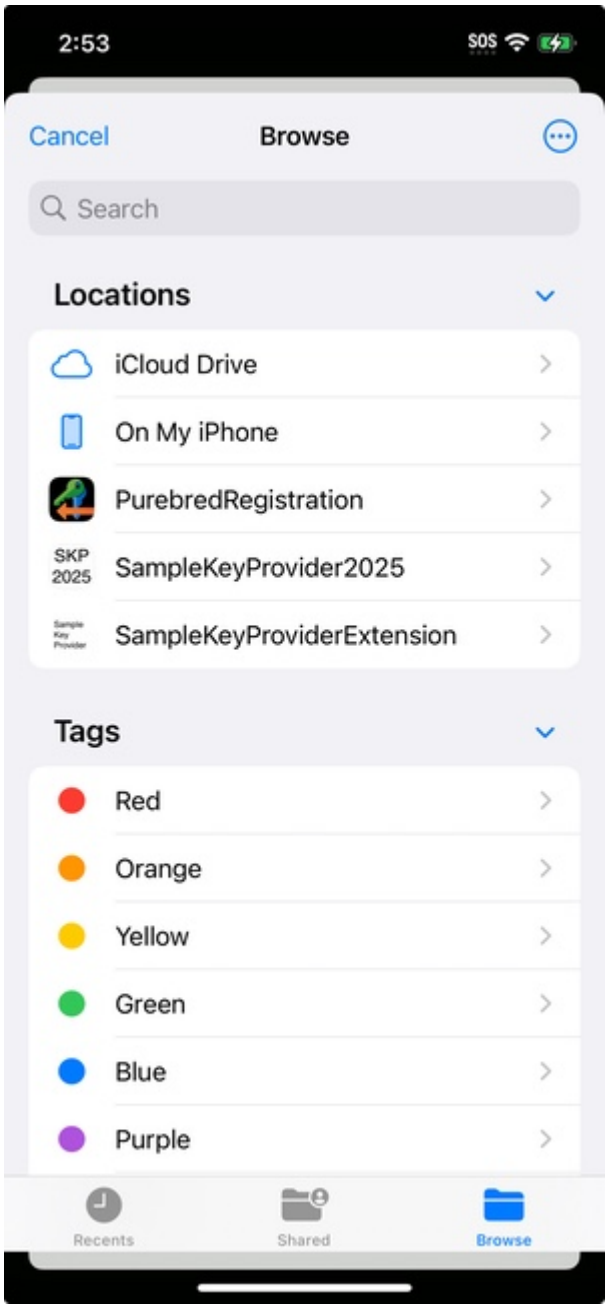
This occurs in the `Key Share Consumer 2025` app in the click handler for the Import Key button. Because `Key Share Consumer 2025` uses Swift UI and file provider APIs use UIKit, a wrapper struct that implements UIViewControllerRepresentable is necessary. The actual instantiation of a `UIDocumentPickerViewController` occurs in the makeUIViewController implementation, as shown below.

```
let docPicker = UIDocumentPickerViewController(forOpeningContentTypes:
utis)
docPicker.delegate = context.coordinator
return docPicker
```

This causes the operating system to display a view that lists sources from which keys may be imported or the most recently used source, if the mechanism has been used previously.

2) User selects provider

The user chooses a source, which may initially be implicit as a previously used provider is displayed by default. The screenshot below shows an example of a list of providers presented for user selection. The appearance is the same as with Legacy Key Sharing.



Provider Selection in Key Sharing 2025

3) Enumerate items

After the user makes a selection, the operating system invokes the subclass of NSFileProviderExtension in the file provider extension. This is a significant difference vs. Legacy Key Sharing. The operating system retrieves an enumerator from the extension, and enumerates over the list of keys made available by the extension. The operating system handles the user interface and the file provider is not provided a list of UTIs desired by the consuming application.

In the `Sample Key Provider 2025` app, enumeration of items is handled by the FileProviderEnumerator class, which is a subclass of NSFileProviderEnumerator.

Because the file provider extension is not presented a list of UTIs that are desired by the consuming application, a set of PKCS #12 files, zip files, and folders are returned in an attempt to map onto the set of custom UTIs used by legacy key sharing. The following table describes the current mapping. Note, because the UTIs conform to different types in legacy vs. 2025, the old UTIs were altered to feature a 2025 suffix to

the first element. Legacy key sharing and key sharing 2025 are thus not compatible in terms of APIs nor UTIs. The only Apple UTI used directly in legacy key sharing was replicated as purebred2025.rsa.pkcs-12, to keep all UTIs distinct across mechanisms.

UTI	Item in Key Sharing 2025	Parent
purebred2025.rsa.pkcs-12	PKCS #12 file	Root
purebred2025.select.all	Folder	Root
purebred2025.select.all_user	Folder	Root
purebred2025.select.all-user	Folder	Root
purebred2025.select.device	Folder	Root
purebred2025.select.signature	Folder	Root
purebred2025.select.encryption	Folder	Root
purebred2025.select.authentication	Folder	Root
purebred2025.select.no_filter	Folder	Root
purebred2025.select.no-filter	Folder	Root
purebred2025.zip.all	Zip file	Root
purebred2025.zip.all_user	Zip file	Root
purebred2025.zip.all-user	Zip file	Root
purebred2025.zip.device	Zip file	Root
purebred2025.zip.signature	Zip file	Root
purebred2025.zip.encryption	Zip file	Root
purebred2025.zip.authentication	Zip file	Root
purebred2025.zip.no_filter	Zip file	Root
purebred2025.zip.no-filter	Zip file	Root

In addition to the UTIs defined for legacy key sharing, an additional UTI is defined for each purebred.select type. The new type is named by appending "-p12" as a suffix. These types represent PKCS #12 files that appear within a folder associated with a purebred.select type, as shown in the following table. These UTIs are new and are necessary to limit when the contents for the folders are enabled for importing.

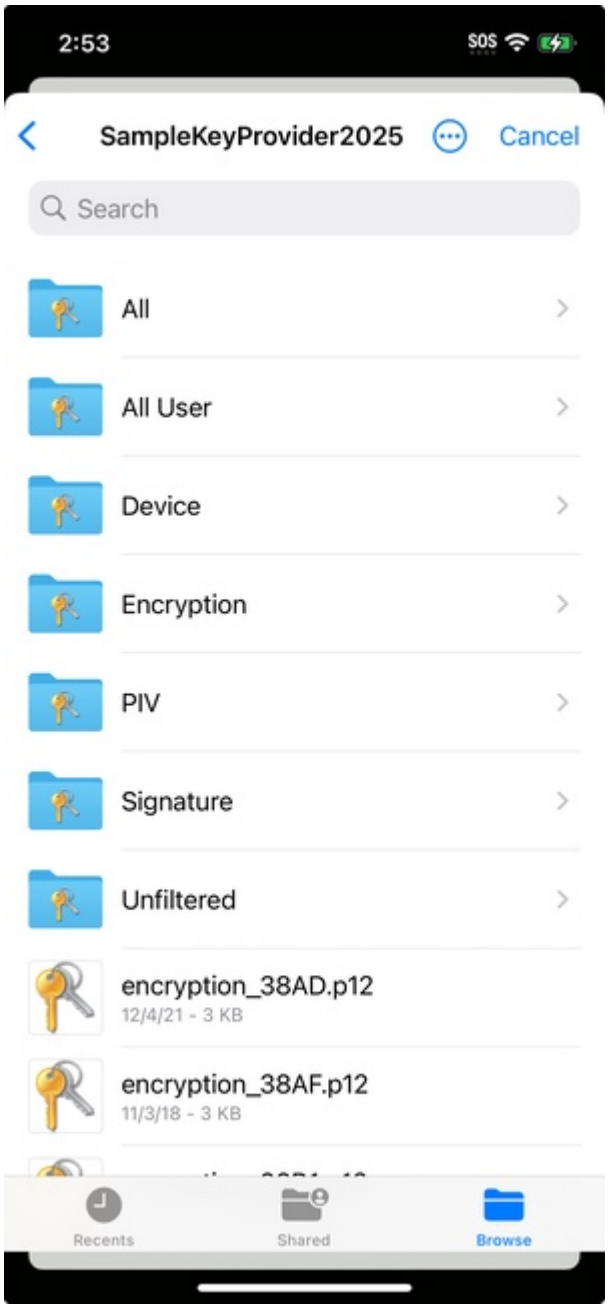
UTI	Item in Key Sharing 2025	Parent
purebred2025.select.all-p12	PKCS #12 file	All folder
purebred2025.select.all_user-p12	PKCS #12 file	All User folder
purebred2025.select.all-user-p12	PKCS #12 file	All User folder

UTI	Item in Key Sharing 2025	Parent
purebred2025.select.device-p12	PKCS #12 file	Device folder
purebred2025.select.signature-p12	PKCS #12 file	Signature folder
purebred2025.select.encryption-p12	PKCS #12 file	Encryption folder
purebred2025.select.authentication-p12	PKCS #12 file	PIV folder
purebred2025.select.no_filter-p12	PKCS #12 file	Unfiltered folder
purebred2025.select.no-filter-p12	PKCS #12 file	Unfiltered folder

This arrangement was defined to account for the fact that the file provider extension does not know what UTIs were requested and the operating system always displays all items with items corresponding to UTIs that were not requested grayed out.

The file provider manages three types of objects: PKCS #12 files, zip files, and folders. PKCS #12 files with the root as a parent are named with the value from the kSecAttrLabel attribute (for Purebred, the value is a SHA-256 hash of the certificate). PKCS #12 files that have a parent other than the root are named using the same identifier but with the folder name prepended followed by a dot, i.e., ".". Folders are named with the name of the folder. Zip files are named with the name of the folder with ".zip" appended. PKCS #12 files are named using the serial number of the certificate appended to an indication of the type of certificate. The creation date associated with PKCS #12 files is the notBefore value from the certificate. The current system time is used for zip files.

After enumerating the items from the provider, the operating system displays the items for user review and selection. The screenshot below shows an example.



Key Display in Key Sharing 2025

One reason the UTIs are distinct between legacy and 2025 is that the definitiond are different by necessity. Here is the `purebred.select.all.user` from legacy key sharing.

```
<dict>
    <key>UTTypeIdentifier</key>
    <string>purebred.select.all.user</string>
    <key>UTTypeConformsTo</key>
    <array>
        <string>public.data</string>
    </array>
    <key>UTTypeDescription</key>
    <string>Poprietary Archive Document</string>
</dict>
```

Here is the definition of `purebred2025.select.all.user` from 2025.

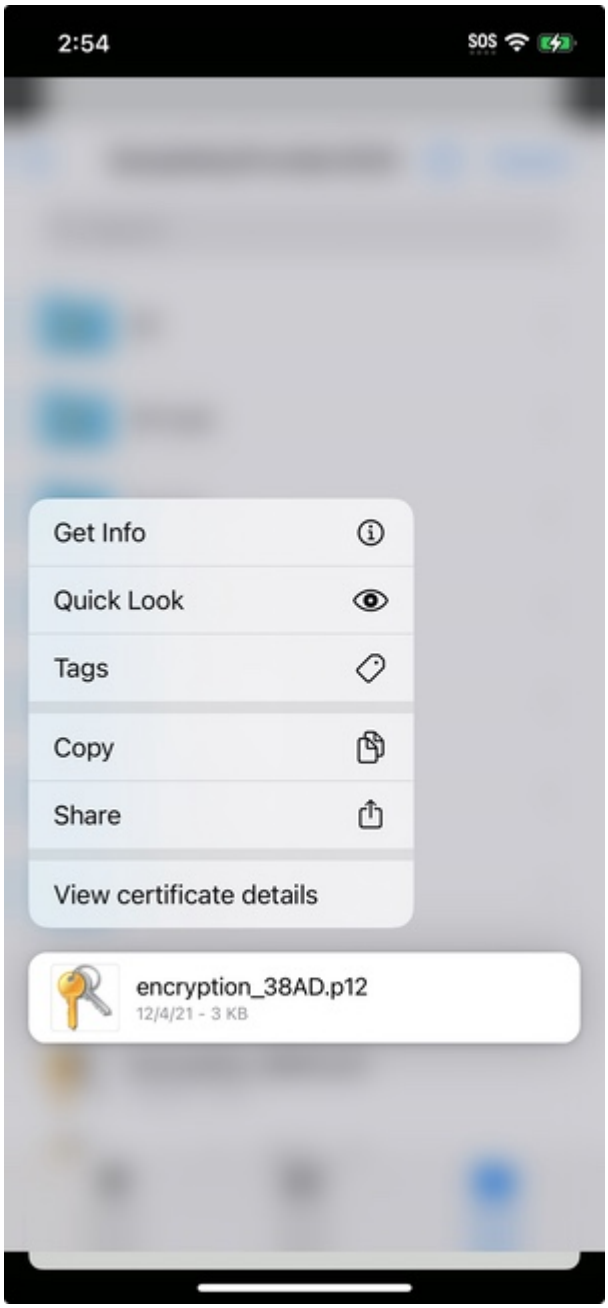
```
<dict>
    <key>UTTypeIdentifier</key>
    <string>purebred2025.select.all.user</string>
    <key>UTTypeConformsTo</key>
    <array>
        <string>public.folder</string>
    </array>
    <key>UTTypeDescription</key>
    <string>X.509 Certificate for User</string>
</dict>
```

The legacy definition conforms to `public.data`, which enables importing PKCS #12 objects. The 2025 definition conforms to `public.folder`, to allow for grouping files by UTI. The new UTIs with the "-p12" suffix are more analogous to the corresponding legacy UTI definition, as shown below, but with conformance to `com.rsa.pkcs-12` instead of `public.data`.

```
<dict>
    <key>UTTypeIdentifier</key>
    <string>purebred2025.select.all.user-p12</string>
    <key>UTTypeConformsTo</key>
    <array>
        <string>com.rsa.pkcs-12</string>
    </array>
    <key>UTTypeDescription</key>
    <string>X.509 Certificate for User</string>
</dict>
```

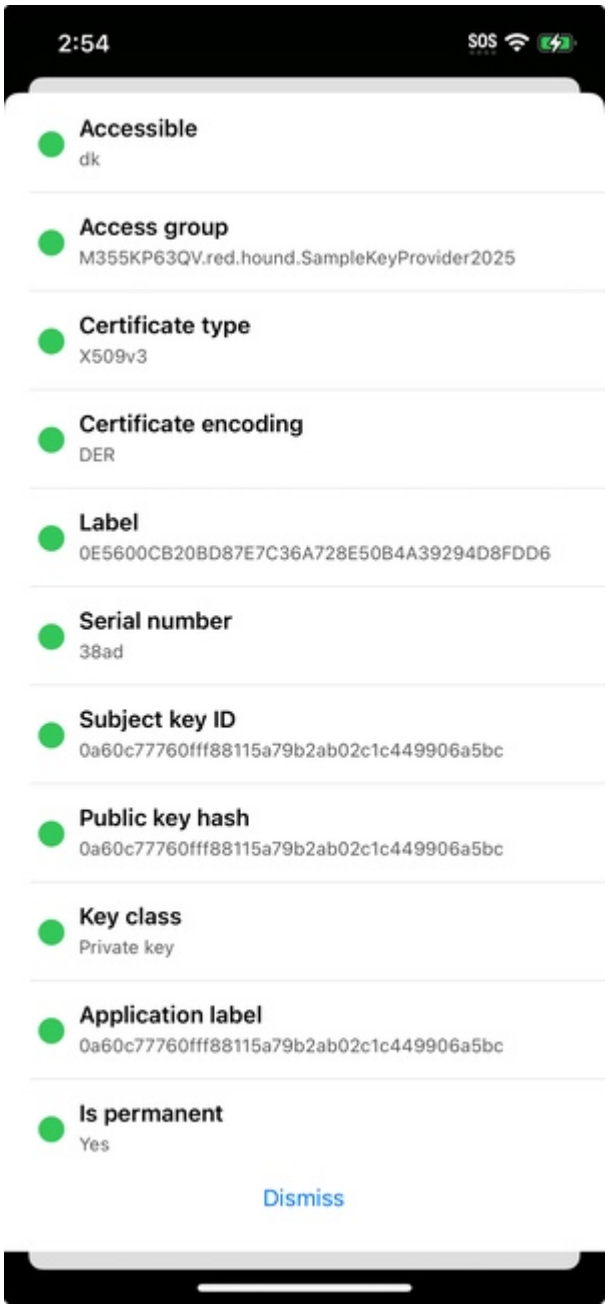
4) Optionally display key details

Where the file provider user interface extension played a primary role in legacy key sharing, in key sharing 2025 it is an optionally used component that simply displays key chain attributes for a selected key. It is invoked when a user long presses an item, as shown below.



Context Menu to Access Key Details in Key Sharing 2025

The screen shot below shows an example of the details view when the user selects the `View Certificate Details` menu item.



Key Details in Key Sharing 2025

5) User selects key to import

Unlike with legacy key sharing, the user selection is made using a user interface that is displayed and managed by the operating system. The interface is similar to the Files app and should be familiar to most users. A selection is made by simply tapping an item. The screenshot in Section 3 provides an example of the view displayed to the user to select a key to import.

6) Get or set password for shared PKCS #12 objects

In the Key Sharing 2025 implementation, the file provider user interface generates a random password the first time a PKCS #12 object is exported then saves that password in the system key chain to facilitate sharing via the key sharing password service and for use in exporting PKCS #12 objects in the future.

7) Return PKCS #12 data

PKCS #12 data is returned via the startProvidingItem method of the NSFileProviderExtension subclass. The data is written to the provided URL. The URL is returned by the file provider extension's urlForItem implementation, which arrives at the URL by appending the identifier to the `documentStorageURL` value read from an instance of NSFileProviderManager.

8.a) Retrieve password

The file provider used in the key sharing 2025 implementation is not permitted to access the system pasteboard. To allow passwords to be returned from the Purebred app to a consuming application, an subclass of NSFileProviderServiceSource was prepared and a custom protocol, KeySharingPassword, was defined. The protocol definition is as shown below.

```
typealias PasswordHandler = (_ password: String?, _ error: NSError?) ->
Void

let keySharingPasswordv1 =
NSFileProviderServiceName("red.hound.KeySharingPassword-v1.0.0")

@objc protocol KeySharingPassword {
    func fetchPassword(_ completionHandler: PasswordHandler?)
}
```

8.b) Read password for shared PKCS #12 objects

The consuming application connects to the service exposed by the file provider extension and uses the `KeySharingPassword` protocol to retrieve the password.

The Key Share Consumer 2025 usage of the service is shown here.