Article

Upgrading Account Security With an Account Authentication Modification Extension

Automatically and transparently convert accounts to Sign in with Apple or to use strong passwords for improved security.

Overview

When accounts have weak, easily guessed passwords, or if an account is breached, the iOS Security Recommendations help users update account passwords. Adding an Account Authentication Modification Extension to your app lets users easily switch to a strong, complex password, or upgrade to Sign in with Apple. The experience for users is simple and painless, often only requiring the tap of a button.

You configure your extension to support upgrading to strong passwords, upgrading to Sign in with Apple, or both. Users initiate upgrades from the following places:

- In Settings > Passwords, when viewing a security recommendation.
- In your app, when the user signs in to their account using a weak or breached password.
- In your app, using a dedicated interface you implement.

If your account security policy requires additional steps, such as two-factor authentication, your extension can include these steps in the upgrade process.

Configure an Associated Domain

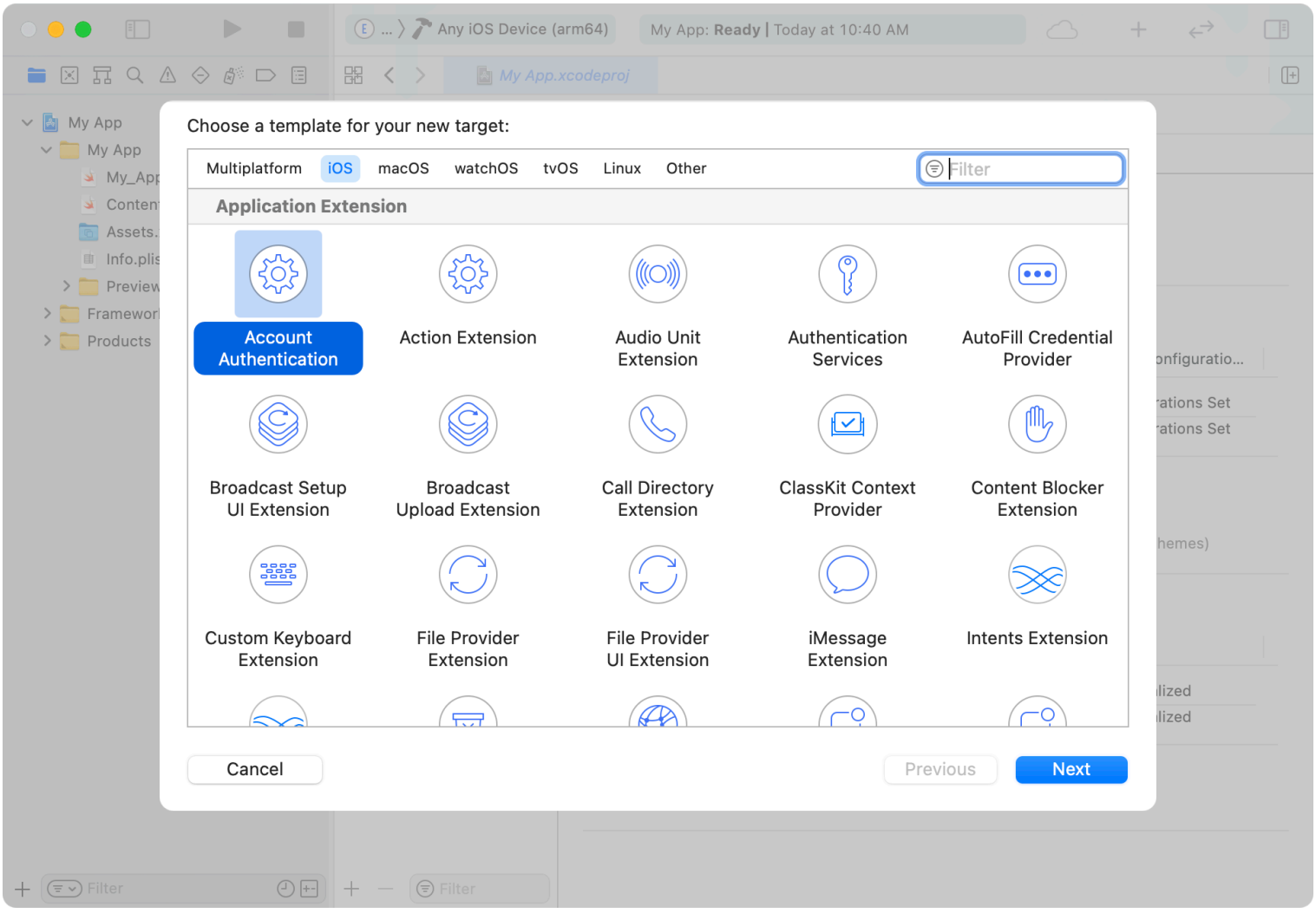
To upgrade user accounts stored in the iOS password manager, you must first establish an association between your app and the domains that to which your account belongs. To make this association, add an associated domain file to your website that includes a `webcredential` entry,

and configure a matching `Associated Domains Entitlement`. For more information about how to configure associated domains, see Supporting associated domains.

Add an Account Authentication Modification Extension to Your App

Xcode provides a starting point to create your extension. To add the Account Authentication Modification extension to your project:

1. Open your app project in Xcode and choose File > New > Target.
2. From the Application Extension group, select Account Authentication, and then click Next.
3. Enter the name of your extension.
4. Click Finish.



Declare Supported Upgrade Types

Your extension's `Info.plist` file specifies the types of upgrades your app supports. To support upgrading weak passwords to strong passwords, set the `ASAccountAuthentication`

ModificationSupportsStrongPasswordChange key to YES. To support upgrading accounts from using passwords to using Sign in with Apple, set the ASAccount AuthenticationModificationSupportsUpgradeToSignInWithApple to YES.

When the system begins a strong password upgrade, it generates a new strong password. If the system-generated password doesn't meet your needs, optionally add the ASAccount AuthenticationModificationPasswordGenerationRequirements key with a string value that describes your requirements. For details about the format of this string, see Customizing Password AutoFill rules. Use the Password Rules Validation Tool to verify the passwords the system generates with your requirements.

Support Automatic Strong Password Upgrades

When the user initiates a strong password upgrade, the system invokes your extension to perform the upgrade. To perform a one-tap automatic upgrade to strong passwords, the system loads your extension and instantiates a subclass of ASAccountAuthenticationModificationView Controller. The system calls the changePasswordWithoutUserInteraction(for: existingCredential:newPassword:userInfo:) method you implement in the view controller.

```
class AccountAuthenticationModificationViewController: ASAccountAuthenticationModifi
    override func changePasswordWithoutUserInteraction(
        for serviceIdentifier: ASCredentialServiceIdentifier,
        existingCredential: ASPasswordCredential,
        newPassword: String,
        userInfo: [AnyHashable : Any]?)
    {
        // Verify with your server that the user is authorized to
        // update the password.

        // Once confirmed, send the user data to your server.
    }
}
```

This method receives the following parameters:

- An ASCredentialServiceIdentifier to identify the service to update, for example a website account.
- The service's current password.
- A system-generated strong password, conforming to ASAccountAuthentication ModificationPasswordGenerationRequirements, if specified.

- A user-info dictionary that contains app-specific information, if your app initiates the request.

To perform the automatic upgrade, the extension communicates with your server to authorize and perform the upgrade. Use the current password, existing token, or credential information from your app to confirm with your server that the user is authorized to perform the password upgrade. Once the server confirms the user authorization, send the new password to your server to perform the upgrade. If the system-generated password doesn't satisfy your server's requirements, you can use a different password than the one the system provides.

After completing the password change on your server, call `completeChangePasswordRequest(updatedCredential:userInfo:)` on the view controller's `extensionContext`, passing the new credential for the service. The system stores the new credential in the Keychain, and removes the old credential.

```
let newCredential = ASPasswordCredential(
    user: existingCredential.user,
    password: newPassword
)
self.extensionContext.completeChangePasswordRequest(updatedCredential: newCredential
```

If the user isn't authorized to perform the upgrade, or the update fails, cancel the request as follows:

```
let error = ASExtensionError(.failed)
self.extensionContext.cancelRequest(withError: error)
```

To display a custom error message to the user, initialize an `ASExtensionError` with `ASExtensionError.Code.failed` and specify an error string, as follows:

```
let error = ASExtensionError(.failed, userInfo: [
    ASExtensionLocalizedFailureReasonErrorKey: "Authentication failed."
])
self.extensionContext.cancelRequest(withError: error)
```

Support Automatic Upgrades to Sign in with Apple

The steps to implement automatic upgrades to Sign in with Apple are very similar to the ones to upgrade to a strong password. To support upgrading to Sign in with Apple, your app must support Sign in with Apple. For more information, see Sign in with Apple and Implementing User Authentication with Sign in with Apple.

When the user initiates an upgrade to Sign in with Apple, the system loads your extension, and invokes the `convertAccountToSignInWithAppleWithoutUserInteraction(for: existingCredential:userInfo:)` method of your view controller.

This method receives the following parameters:

- An `ASCredentialServiceIdentifier` that identifies the service to update, for example a website account.
- The service's current password.
- A user-info dictionary that contains app-specific information, if your app initiates the request.

Use the current password to authorize with your server that the user can perform the upgrade. If not, fail the request as described above.

To perform the upgrade, request the Sign in with Apple credentials from the extensionContext by calling `getSignInWithAppleUpgradeAuthorization(state:nonce:completionHandler:)`. If the system provides a Sign in with Apple credential, send the credential information to your server, and then complete the request by calling `completeUpgradeToSignInWithApple(userInfo:)`. Once complete, the system removes the old password from the Keychain.

```
let myState: String = ...
let myNonce: String = ...

self.extensionContext.getSignInWithAppleUpgradeAuthorization(
    state: myState,
    nonce: myNonce
) { appleIDCredential, error in
    guard let appleIDCredential = appleIDCredential else {
        self.extensionContext.cancelRequest(
            withError: ASExtensionError(.failed)
        )
        return
    }

    // Verify with your server that the user is authorized to
    // convert the account to Sign in with Apple.

    let userIdentifier = appleIDCredential.user
    let fullName = appleIDCredential.fullName
    let email = appleIDCredential.email
    // Once confirmed, send the user data to your server.

    self.extensionContext.completeUpgradeToSignInWithApple()
```

```
}
```

To create the Sign in with Apple credential, the system prompts the user and communicates with Apple. If the user cancels the request, the system calls the completion handler with an error that specifies a `ASExtensionError.Code.userCanceled` code. In that case, cancel the request using the same error code and no failure reason, and don't show any additional user interface. If the device can't communicate with Apple's servers, the completion handler may be passed `nil` credentials. In this case, cancel the request with an error that specifies `ASExtensionError.Code.failed` as the error code.

Perform Upgrades With Additional Security Requirements

If the user initiates an account authentication upgrade but you require additional security steps, such as using two-factor authentication, fail the initial automatic request with an error that specifies user interaction is required.

```
let error = ASExtensionError(.userInteractionRequired)
self.extensionContext.cancelRequest(withError: error)
```

The system initiates a new request that includes presenting the view controller's view by calling either `prepareInterfaceToChangePassword(for:existingCredential:newPassword:userInfo:)` or `prepareInterfaceToConvertAccountToSignInWithApple(for:existingCredential:userInfo:)`, depending on the type of the original request. Configure your view controller's view appropriately for the request and return from the method.

After your method returns, the system presents your view controller's view with a system-provided navigation bar, including a cancel button. Once the system presents the view, guide the user through performing the additional steps. When the user completes the steps, continue the upgrade process. For a strong password upgrade, commit the new strong password, provided in the original request, to your server. For an upgrade to Sign in with Apple, call `getSignInWithAppleUpgradeAuthorization(state:nonce:completionHandler:)` to request a new Sign in with Apple credential as described earlier.

If the user fails to complete the additional steps, or the upgrade fails for any other reason, call `cancelRequest(withError:)` to cancel the request as described in earlier examples. The system dismisses the view controller's view.

If the user taps the system-provided cancel button, the system dismisses the interface and calls your view controller's `cancelRequest()` method. If you need to perform any cleanup work, override this method and call the superclass `cancelRequest()` method to complete the process.

Initiate Security Upgrades From Within Your App

To initiate upgrades to strong passwords or to use Sign in with Apple from within your app, use the `ASAccountAuthenticationModificationController` class.

When your app initiates an upgrade request, the work your extension does is the same as the previous sections describe. To configure the request, instantiate either an `ASAccountAuthenticationModificationUpgradePasswordToStrongPasswordRequest` or `ASAccountAuthenticationModificationReplacePasswordWithSignInWithAppleRequest` object, passing the following parameters:

- The user name for the user's account.
- An `ASCredentialServiceIdentifier` to identify the service to update, for example a website account.
- A user info dictionary that contains any authentication information the extension needs.

Unlike the system-initiated upgrades, your extension doesn't receive an existing password in the calls to `changePasswordWithoutUserInteraction(for:existingCredential:newPassword:userInfo:)` or `convertAccountToSignInWithAppleWithoutUserInteraction(for:existingCredential:userInfo:)`. If your extension needs existing password or authentication information to perform the authorization with your server, include it in the user info dictionary.

To perform the request:

1. Instantiate an `ASAccountAuthenticationModificationController` object.
2. Set the controller's delegate property to an object that conforms to the `ASAccountAuthenticationModificationControllerDelegate` protocol.
3. Set the controller's `presentationContextProvider` property to an object that conforms to the `ASAccountAuthenticationModificationControllerPresentationContextProviding` protocol.
4. Call `perform(_:)` on the controller to initiate the request.

The following example shows how to configure and initiate a request to upgrade to a strong password:

```
let username = "jappleseed"
let serviceIdentifier = ASCredentialServiceIdentifier(
    identifier: "myservice.example.com",
    type: .domain
)

let userInfo = ["com.example.authTokenKey": authToken]
```

```
let upgradeRequest = ASAccountAuthenticationModificationUpgradePasswordToStrongPassw
    user: username,
    serviceIdentifier: serviceIdentifier,
    userInfo: userInfo
)

let requestController = ASAccountAuthenticationModificationController()
requestController.delegate = self
requestController.presentationContextProvider = self
requestController.perform(upgradeRequest)
```

For upgrading to Sign in with Apple, the process is the same, except you instantiate an `ASAccountAuthenticationModificationReplacePasswordWithSignInWithAppleRequest` object instead:

```
let upgradeRequest = ASAccountAuthenticationModificationReplacePasswordWithSignInWi
    user: username,
    serviceIdentifier: serviceIdentifier,
    userInfo: userInfo
)
```

When the request completes, the controller calls either `accountAuthenticationModificationController(_:didSuccessfullyComplete:userInfo:)` or `accountAuthenticationModificationController(_:didFail:error:)`, depending on result of the upgrade. Inform the user of the results accordingly.

See Also

Automatic security upgrades

`class` `ASAccountAuthenticationModificationController`

An object that performs a request to modify an account's authentication properties.

`class` `ASAccountAuthenticationModificationViewController`

A view controller that can upgrade user passwords to strong passwords, or convert accounts to use Sign in with Apple.

`class` `ASAccountAuthenticationModificationExtensionContext`

An object that you interact with to change an account's password or to upgrade to Sign in with Apple.