BlockChain Commons Security Implementation Review (revision 3)

Evaluated application version: 1.5.0

LocalAuthentication Policy

SeedTool mediates access to sensitive operations using a LockRevealButton, which conditionally displays content and/or controls based upon evaluation of the `.deviceOwnerAuthentication` LocalAuthentication policy.

Under this policy, biometrics enrolment is optional and challenges may be declined, allowing the person holding the device to authenticate using the device passcode instead.

- On a device with Face ID enrolled, place your thumb over the front-facing camera and sensor "notch". Keep your thumb in place until the final step below.
- Tap on a Seed Detail "Authenticate" button and wait until the "Face Not Recognized" alert appears.
- Tap on "Try Face ID Again" and wait until the "Face Not Recognized" message reappears.
- Remove your thumb from the sensor notch and tap on "Enter Passcode" to complete the authentication.

Does product documentation adequately describe the role of biometrics?

The "What is a Seed?" section of the application documentation clearly explains that "*seeds cannot be accessed without your device passcode or biometric authorization*" (link).

However, other statements might be allowing users to assume that a biometrics challenge in SeedTool is standalone, with no fallback offer to authenticate using the device passcode.

From `README.md` (link):

> Seed Tool uses 2FA by combining an Apple login with biometric authentication.

From `about-seed-tool.md` (link):

> Gordian Seed Tool […] protects them with 2FA and biometrics […]

Use of SeedTool is unlikely to correct the user's assumption, because the the presence of (or lack of) a fallback is only revealed after the *second* LocalAuthentication biometrics failure.

The point here is not to (pedantically) highlight an inaccuracy in the documentation. A hypothetical user thinking about whether authentication is biometrics-only might be looking to overcome device/passcode integrity issues (e.g.: coercive domestic violence situations).

For the sake of that user, onscreen messaging and app documentation should be very clear about the role of biometrics and the passcode. Some users might also value the ability to choose between `.deviceOwnerAuthentication` and `.deviceOwnerAuthenticationWithBiometrics` for their seeds.

There's no authoritative expression of the notion that the user has been authenticated. Is this intentional?

Each LockRevealButton requires its own separate successful authentication in order to display "unlocked" content, but the underlying `Authentication` object appears to be managing state that is/was intended to be shared.

Should "unlocked" content be re-locked when the application transitions to the background?

Even if leaving the content unlocked is entirely benign, I believe users would see re-locking as symmetrical and reassuring.

Should QR codes be occluded for iOS app-switcher snapshots?

This is generally expected of applications displaying sensitive content.

Note: if you choose to implement this behaviour, SwiftUI `.privacySensitive()` content won't be redacted when it's in a modally-presented view under iOS/iPadOS 15.

Why do simulator and DEBUG builds always bypass the LocalAuthentication check?

The simulator supports LocalAuthentication challenges and DEBUG builds also run on physical devices, so the conditional compilation in `Authentication.swift` (link) seems overly broad.

Could the scope of the authentication bypass be reduced, perhaps by conditionally compiling an early guard-else that checks for a POSIX environment value or a command line argument?

Local Storage and iCloud Syncing

During the initial SeedTool evaluation, I noted that local-storage save errors aren't reported to the user, and so I expected a more detailed review would reveal inconsistencies in the reporting of partial failures.

That doesn't seem to be the case at all. From what I can see in the `Saveable`, `Model`, and `Cloud` types and after scanning through project occurrences of `catch` and `.failure`, all save/sync errors are treated as transient and irrelevant to the user. Aside from recording some errors in OSLog messages, the pervasive reaction to an error is to silently abandon the affected operation.

This is not an approach I would have chosen, but there's little benefit in speculating about better options if it has been implemented according to spec and is known to be working as-intended in version 1.5.0.

If there are concerns about reliability, we can return to the remaining time for reviewing this code once we know what those concerns are.

Remote Notification Processing and PassthroughSubject

The SeedTool AppDelegate implementation of `application(_:didReceiveRemoteNotification:fetchCompletionHandler:)` passes along the responsibility for processing a notification and calling the fetchCompletionHandler by way of a PassthroughSubject.

Applications are obligated to call the fetch completion handler exactly-once, but PassthroughSubject sends input events based on subscription demand. To ensure the completion handler is called exactly-once you must ensure there is exactly-one subscriber at the moment `send(_:)` is called.

For this particular PassthroughSubject, subscribers are created by a SwiftUI `onReceive` in the SeedToolApp `body`, making the number of subscribers at any moment sensitive to things like multi-scene support and the SwiftUI render loop. Is this the right abstraction for the job?