Background: streamlining interfaces between ERC-7579 and ERC-6900. Enable plugins to be compatible with both standards in certain cases. Generally, work together to grow what is an exciting, but small and nascent ecosystem around Modular Smart Accounts. Most of the conversation so far has centered around the permissions model. Regardless of permissions or no permissions, there are other ways to improve both proposals that might be worth pursuing. As such, this document does not touch on the permission system differences which are the main logical differentiator between the two standards.

See: ERC-6900 compatibility with minimal smart account proposal

Summary of Current State

1. `onInstall`, `onUninstall` are already compatible.
2. `isModuleType` doesn't make sense for 6900 to implement, this is the explicit role of the plugin manifest, and not supporting this isn't a blocker for plugins to be simultaneously compatible with both 6900 & 7579.
3. Function ID removal: implemented a version with this: https://github.com/erc6900/reference-implementation/pull/41
    1. This causes an issue with plugin state configuration, when a plugin implements more than one hook function or validation type, internal-only switching can make certain account configs impossible. Full context here: https://github.com/erc6900/reference-implementation/pull/41#issuecomment-1997756363
    2. Equivalent problem found in 7579: Hook destructuring, as done by the ModuleKit contract `ERC7579HookDestruct`, becomes incompatible with different ERC7579 accounts that have additional native functions, handler functions, etc. Context here: https://github.com/rhinestonewtf/modulekit/blob/main/packages/modulekit/src/modules/ERC7579HookDestruct.sol#L54-L56
    3. Question: Right now in 7579, installing a hook breaks composability with functions the hook isn't aware of – either from other plugins or from native functions on account implementations. A function id (or other selector mechanism) can solve this, but can this be handled in a different way?
4. Validator function name change: implemented, but depends on function ID removal to be fully compatible.
5. ERC-1271 support: implemented as a validation function here: https://github.com/erc6900/reference-implementation/pull/42
    1. Question: this seems to be optional in 7579. Is the optional configuration exposed anywhere?

6. Hook renaming: We have a distinction of validation vs. execution hooks, so calling it a “pre execution hook” lets us differentiate between a step in validation and a step in runtime.
    1. Question: Is there work being done on 7579 to support multiple validation steps, just like execution hooks?
7. Pre-hook value param: discussion in the original doc, but leaning towards keeping it for DeFi use cases and due to low implementation costs.

Architecture Review of 7579

General theme

Because ERC-7579 is so focused on granting flexibility to account developers, it doesn’t standardize account behavior enough for plugins to generalize across different account implementations.

No standard way to associate execution with validation

- 7579 doesn’t say anything about how to decide which validators can be used to validate for which calls. So you could have two different 7579-compliant accounts where:
    - one account says that any installed validator can be used to approve any call (validators apply globally across any execution function).
    - the other account has some way of specifying that certain calls require certain validators, so it would be able to have, for example, an owner validator and a session key validator, where certain executions require certain validators.
- The issue is that since 7579 doesn’t prescribe how validation and execution is associated, it’s impossible to write a module (plugin) that requires that a particular execution must be used with a particular validation in a way that works across all account implementations.
- We view this a serious problem for interoperability. Every plugin we’ve written so far (multi-owner, session key, cold storage), has needed to associate validation with execution as part of its core functionality.
- This cannot be left implementation-dependent per account: a plugin built to work with one account will not work with other accounts.
- Two example scenarios to consider here:
    - A plugin wants to add execution behavior that can only occur after a validation it defines itself, e.g. session key validation.
    - A plugin wants to add execution behavior that can only occur after validation defined by some other plugin, e.g. owner key validation.
- Trying to standardize a way that a plugin can specify each of these likely leads down a path similar to the execution function and dependency sections of the 6900 manifest.

No standard data format passed to hooks

- A “hook” plugin gets to define a `preCheck` hook which is passed the `msgData` sent to the smart account. But the standard leaves the structure of this data implementation-dependent per account, where the data is presumably somehow indicating which execution module should do something and encoding the data for that module. Thus, a hook module must be written for one particular interpretation of the calldata and is tied to a particular account implementation.

No validation hooks

- This makes implementing general-purpose gas limits difficult or impossible. Accounts that don’t carefully limit gas spending are highly vulnerable to having their funds drained if there’s any op that an attacker is able to validate. Under 7579, every validation function needs to build its own defenses against this.
- It’s possible the intention is that you can add multiple validations to something and require that they all pass, basically treating validation like a validation hook. If so, there is no standardized means of doing this, which again means that a plugin that works for one account will not work for others.
- To be fair, general-purpose gas limits aren’t fully possible in 6900 at the moment either, but we’re a lot closer (we just need to add validation hooks that apply to any execution function).

No standard way for only certain hooks to run

- A plugin author might want their hooks to only run around a certain call, but there is no standard way for a hook to indicate this. That means that there’s no better standard mechanism than all hooks running on all calls (and each hook doing nothing if it doesn’t apply to that call), which can waste gas. While an account can implement its own way of allowing hooks to configure this, a hook built around one such mechanism won’t work with others.

Questions for 7579 Authors

- Right now in 7579, installing a hook breaks composability with functions the hook isn't aware of – either from other plugins or from native functions on account implementations. A function id (or other selector mechanism) can solve this, but can this be handled in a different way?
- ERC-1271 seems to be optional in 7579. Is the optional configuration exposed anywhere?
- Is there work being done on 7579 to support multiple validation steps, just like execution hooks?
- It doesn’t seem like 7579 standardizes the data encoding for execute with mode. It is implemented in one particular way in the 7579 reference implementation – is that normative?
- The function `supportsAccountMode` actually deals with execution mode configs, should this be renamed?