Not sure if something like this has already been discussed, but I would like to propose the following Dirt Simple (no BLS verification, very easy to audit and formally verify) Withdrawal Contract:

There are only two functions. One is a constant function named “is_withdrawal_active” which returns bool FALSE.

The second is a function “set_withdrawal_address” which takes arguments “(address withdrawal_address_to_set, bool is_final, bytes validator_pubkey, bytes bls_signature)”, emits the submitted arguments as an event, and then throws them away with no further processing.

The contract should be written in and compiled from whichever language will make it easiest to audit and formally verify. An important goal of this design is to allow that to happen as soon as conceivably possible, in order to support use cases like trustless staking pools.

In order to set a withdrawal address, the address and the is_final bit are signed with the withdrawal

key for a single validator ID. Then the address, the bit, the pubkey for the corresponding validator

, and the created signature are submitted via the “set_withdrawal_address” function. The rule for interpreting the Dirt Simple Withdrawal Contract in the future is that the withdrawal address for each validator is set to:

1. If no valid BLS signed message for the validator has ever been sent to the Withdrawal Contract, deposits will eventually be claimable by signed withdrawal message from the Withdrawal Key.
2. If any valid BLS message for the validator has ever been submitted with the is_final bit set to TRUE, then the deposit will be sent to the address given in the first valid message which was submitted with an is_final bit set to TRUE. All others will be ignored.
3. If any valid BLS message has ever been submitted for the validator, but all valid messages have set the is_final bit to FALSE, then the deposit will be sent to the address given in the latest message submitted. All others will be ignored.

Note that since the contract itself neither verifies BLS signatures nor keeps track of either final or updateable messages, all such verification must be done out of band.

If desired in a future hard fork, a more complex version of the Withdrawal Contract may be deployed in place of the Dirt Simple Withdrawal Contract, which may do things such as actually verifying the BLS signatures, maintaining an actual list of current withdrawal addresses which have been set, etc.

Eventually, when withdrawals are ready to be enabled, the Dirt Simple Withdrawal Contract or its later replacements will be updated so that the “is_withdrawal_active” function returns TRUE and deposits will actually be processed for validators which have been exited within the beacon chain. It is up to the specifications of later phases to decide what that withdrawal logic is and how it is implemented.

At the time of deploying the Dirt Simple Withdrawal Contract there would be only a very basic EIP created which specifies the rules for updating withdrawal addresses and specifies that withdrawals to addresses set in this way will be performed as simple value sends with no attached metadata. If various contract designers want to use such information when standards for it are eventually decided upon, they will have to design updateable contracts which can retrieve any necessary metadata from, e.g. a beacon chain light client within the EVM when it is standardised and released. Similarly, contract designers who anticipate needing to differentiate based on the specifics of particular validator IDs/histories should be careful to deploy different

upgradeable contracts for each validator, since there would be no way to distinguish the source of an incoming deposit.

Note that the above Dirt Simple Withdrawal contract is sufficient both

for setting a contract in charge of any validator income and

for enabling transfers or tokenisation of exited deposits prior to the existence of actual withdrawal or transfer capability. To irrevocably commit to a contract owning exited funds one simply sends a signed message with the is_final bit set to the Dirt Simple Withdrawal Contract. If desired this can be done prior to depositing in the deposit contract (with deposit data specifying the same withdrawal key which was used to set the withdrawal address for that validator). To transfer or tokenise funds which have already been exited one simply uses an appropriate contract in this fashion. Several audited examples already exist with the necessary functionality from the world of DeFi.

It’s my belief that this would be the fastest and safest path for enabling trustless staking pools and trading or tokenisation of exited validator balances. It is backwards compatible with all existing mainnet validators and requires very minimal due diligence and associated tooling. It would also allow projects which need to develop and audit contracts for trustless staking pools to begin that development immediately, even before the Dirt Simple Withdrawal Contract was successfully audited, verified, and deployed. I’m open to feedback and improvements to the design. One possible improvement, for example, would be to design the signature format in a way that would make it amenable to easy aggregation. But the basic philosophy here is that the Dirt Simple Withdrawal Contract has no logic, only a specification for the future interpretation of submitted messages.