Simple eth1 withdrawals (beacon-chain centric)

[This is a followup and alternative proposal to @technocrypto's Dirt Simple Withdrawal Contract]

This proposal is a beacon-chain-centric withdrawal plan (a place where we have native access to validators and to BLS verification) that keeps the eth1 promise to an absolute minimum.

This allows for simple (but expressive) withdrawal contracts to be written today and for eth1 addr as withdrawal credentials to be submit as validator deposits today (well as soon as we write it in the spec's repo). More sophisticated logic/accesses to beacon state cannot

be carried out with this proposal, but (1) I suspect that 80%+ of withdrawal contract use cases will be satisfied and (2) additional use cases will be opened up when eth1-native reads to beacon state (post-merge) are spec'd/implemented.

Note, the core commitment to getting this going would be to Eth1AddressPrefix

withdrawal credentials and to the fact that withdrawn ETH will end up in the specified withdrawal address as a simple send

. All other details are for illustration that this idea is end-to-end coherent.

Edit

: @vbuterin pointed out that we don't need a new withdrawal_address

field on Validator

. Instead the BLSSetWithdrawalAddress

would change the withdrawal_credentials

from BLS withdrawal credentials to Eth1 withdrawal credentials. Thanks!

tl;dr

Beacon chain changes

:

- Specify Eth1AddressPrefix

– 0x01

– withdrawal prefix that allows to specifies the last 20 bytes of the credentials to be a 20-byte eth1-addr

- Validators will have an withdrawal_address

. This is initiated to ZERO_ADDRESS

- 0x00..00
- When withdrawable
- if prefix == BLS_WITHDRAWAL_PREFIX

: * validator must submit a BLSSetWithdrawalAddress

operation to set their withdrawal_address

before they can be withdrawn (this can be sent before withdrawable)

- BLSSetWithdrawalAddress

contains the destination eth1-addr and is signed by the bls withdrawal credentials

- once this withdrawal_address

is set and the validator is withdrawable, it gets withdrawn at next epoch transition

- validator must submit a BLSSetWithdrawalAddress

operation to set their withdrawal_address

before they can be withdrawn (this can be sent before withdrawable)

- BLSSetWithdrawalAddress

contains the destination eth1-addr and is signed by the bls withdrawal credentials

- once this withdrawal_address

is set and the validator is withdrawable, it gets withdrawn at next epoch transition

- if prefix == Eth1AddressPrefix

, validator gets auto-withdrawn at withdrawable epoch during the epoch transition (see process_withdrawals

)

- Being “withdrawn” is the action of a withdrawal receipt being added to beacon state (see withdraw_validator

)

- if prefix == BLS_WITHDRAWAL_PREFIX

: * validator must submit a BLSSetWithdrawalAddress

operation to set their withdrawal_address

before they can be withdrawn (this can be sent before withdrawable)

- BLSSetWithdrawalAddress

contains the destination eth1-addr and is signed by the bls withdrawal credentials

- once this withdrawal_address

is set and the validator is withdrawable, it gets withdrawn at next epoch transition

- validator must submit a BLSSetWithdrawalAddress

operation to set their withdrawal_address

before they can be withdrawn (this can be sent before withdrawable)

- BLSSetWithdrawalAddress

contains the destination eth1-addr and is signed by the bls withdrawal credentials

- once this withdrawal_address

is set and the validator is withdrawable, it gets withdrawn at next epoch transition

- if prefix == Eth1AddressPrefix

, validator gets auto-withdrawn at withdrawable epoch during the epoch transition (see process_withdrawals

)

- Being “withdrawn” is the action of a withdrawal receipt being added to beacon state (see withdraw_validator

)

- Clean up
- clean up withdrawn validator indices after year+
- clean up withdrawal receipts after time/consumed
- clean up withdrawn validator indices after year+
- clean up withdrawal receipts after time/consumed

Eth1 functionality

:

- Eth1 has access to beacon chain receipt merkle root (or entire list)
- Track consumed receipts in either a system contract or some system state
- Normal TX to special system withdrawal contract to finish withdrawal
- Normal sender of TX pays gas. Maybe can pay gas with validator balance if/when AA
- TX hits a system address. If withdrawal receipt not consumed:
- send ETH to withdrawal_address (normal transfer, no data)
- if/when returns to system contract, mark withdrawal receipt as consumed
- send ETH to withdrawal_address (normal transfer, no data)
- if/when returns to system contract, mark withdrawal receipt as consumed
- Normal sender of TX pays gas. Maybe can pay gas with validator balance if/when AA
- TX hits a system address. If withdrawal receipt not consumed:
- send ETH to withdrawal_address (normal transfer, no data)
- if/when returns to system contract, mark withdrawal receipt as consumed
- send ETH to withdrawal_address (normal transfer, no data)
- if/when returns to system contract, mark withdrawal receipt as consumed

Beacon chain changes

- Add withdrawal_address

to validators, initialize all to ZERO_ADDRESS

- Addr withdrawn_epoch

to validators, initialize all to FAR_FUTURE_EPOCH

BLSWithdrawalPrefix

- Add new beacon operation BLSSetWithdrawalAddress
- Allows user with an BLSWithdrawalPrefix

withdrawal prefix to set their eth1 withdrawal address via a signed message with their BLS withdrawal keys

- Can be performed any time before withdrawn
- Options for discussion:
- Should operation be able to be performed more than once?
- If so, should it have a is_final

feature like in Dirt Simple Withdrawal Contract?

- Probably would need a fee structure to get these included on chain. If so, need to consider implications of transferring balances of active validators (security). Simplest is to only allow for exited+ validators.
- Should operation be able to be performed more than once?
- If so, should it have a is_final

feature like in Dirt Simple Withdrawal Contract?

- Probably would need a fee structure to get these included on chain. If so, need to consider implications of transferring balances of active validators (security). Simplest is to only allow for exited+ validators.
- Allows user with an BLSWithdrawalPrefix

withdrawal prefix to set their eth1 withdrawal address via a signed message with their BLS withdrawal keys

- Can be performed any time before withdrawn
- Options for discussion:
- Should operation be able to be performed more than once?
- If so, should it have a is_final

feature like in Dirt Simple Withdrawal Contract?

- Probably would need a fee structure to get these included on chain. If so, need to consider implications of transferring balances of active validators (security). Simplest is to only allow for exited+ validators.
- Should operation be able to be performed more than once?
- If so, should it have a is_final

feature like in Dirt Simple Withdrawal Contract?

- Probably would need a fee structure to get these included on chain. If so, need to consider implications of transferring balances of active validators (security). Simplest is to only allow for exited+ validators.
- Sample code:

class SignedBLSSetWithdrawalAddress(Container): message: BLSSetWithdrawalAddress signature: BLSSignature

class BLSSetWithdrawalAddress(Container): validator_index: ValidatorIndex credentials_pubkey: BLSPubkey withdrawal_address: Eth1Address

def process_bls_set_withdrawal_address(state: BeaconState, signed_bls_set_withdrawal_address: BLSSetWithdrawalAddress) -> None: set_withdrawal_address = signed_bls_set_withdrawal_address.message validator = state.validators[set_withdrawal_address.validator_index] # Verify the validator is not already withdrawn assert not is_withdrawn_validator(validator, get_current_epoch(state)) # Verify that the validator has BLS withdrawal credentials assert validator.withdrawal_credentials[0] == BLSWithdrawalPrefix # Verify that the pubkey is the correct pubkey behind the credentials assert hash(set_withdrawal_address.credentials_pubkey)[1:] == validator.withdrawal_credentials[1:] # Maybe verify if withdrawal_address == ZERO_ADDRESS to only allow setting once # Verify signature domain = get_domain(state, DOMAIN_BLS_SET_WITHDRAWAL_ADDRESS, get_current_epoch(state)) signing_root = compute_signing_root(set_withdrawal_address, domain) assert bls.Verify(set_withdrawal_address.credentials_pubkey, signing_root, signed_bls_set_withdrawal_address.signature) # Set eth1 withdrawal address validator.withdrawal_address = set_withdrawal_address.withdrawal_address

Withdraw validators at epoch boundary

Add process_withdrawals

to process_epoch

. Note that Eth1AddressPrefix

withdrawals happen automatically, while BLSWithdrawalPrefix

withdrawals require a BLSSetWithdrawalAddress

before the withdrawal can occur.

def withdraw_validator(state: BeaconState, validator: Validator) -> None: validator.withdrawn_epoch = get_current_epoch(state) # Append withdrawal receipt to the state withdrawal receipts # (structure tbd)

def process_withdrawals(state: BeaconState) -> None: withdrawable_validators = [validator for validator in state.validators if (is_withdrawable_validator(validator, get_current_epoch(state) and not is_withdrawn_validator(validator, get_current_epoch(state)))] for validator in withdrawable_validators: if (validator.withdrawal_credentials[0] == BLSWithdrawalPrefix and validator.withdrawal_address != ZERO_ADDRESS): withdraw_validator(state, validator) elif validator.withdrawal_credentials[0] == Eth1AddressPrefix: validator.withdrawal_address = Eth1Address(validator.withdrawal_credentials[12:]) withdraw_validator(state, validator)

Eth1 functionality

Most of the details are in the above tl;dr. The two core elements are

1. once a withdrawal has been initiated on beacon chain side, a normal user TX must trigger the withdrawal to the eth1 addr

2. from the destination addr's (validator.withdrawal_address

) perspective, it just looks like ETH was transferred to it

The particular details of tracking receipt consumption don't have to be fully figured out in this proposal. If we can agree on the two simple points above, that is enough to specify the Eth1AddressPrefix

withdrawal credentials and build/deploy withdrawal contracts.

There are potentially more sophisticated features that cannot

be built with the simple scheme until beacon chain reads are implemented, but I would argue that most designs can be accomplished.

Two important notes when considering designs:

1. Use of child withdrawal contracts can make the handling of withdrawals more granular for a larger pool

2. One interesting thing people want to know is if a withdrawn validator is_slashed

. Yes, your pool might be able to do more punitive things but if you use child contracts for granular withdrawal processing, the eth2 protocol will already have punished the slashed validator through the burning of some amount of ETH, so this baseline punitiveness along with granular parsing provides a pretty solid baseline functionality.