With the validators participating on the p2p network, it's easy (and trivial if you use Handel) to deduce the mapping between validator's wallet public keys (pk) <–> ip address. Simply track pk's committee membership over time and compare with the ip addresses actually participating. A few epochs are enough to deduce the one-to-one mapping.

Hiding behind Tor is a good solution, but can be complicated at scale. Tor has also been attacked in the past. We may spend more time of Tor-like solutions going forward; AFAWK Mikerah is already working on this.

What we describe here is a solution to render the one-to-one mapping pk-ip completely opaque. Our scheme uses linkable ring signatures, or alternatively a zero knowledge proof (ZKP). The idea is that validators will use a new public key for participating, without revealing the link between the wallet used to stake and this new public key.

Here is the high level mechanism, for a user U who wants to be a validator. The detailed ZKP is below.

a) U burns 32 eth by calling a specific contract.

b) U waits up to three months (the idea is that U waits until enough people have burnt stake.)

c) U registers himself as a validator, with the new pk he will be using as a validator, by sending a transaction with a linkable ring signature.

This does not break slashing:

- rewards are put on the new pk.
- when U exits, 32 eth are deposited to the new pk.
- if U is slashed the amount given back is decreased as defined by the slashing rules.

There are a few subtleties, though:

- The new pk has to be used carefully by the user if he doesn't want to reveal his identity.
- The transaction at step c) cannot pay any fee, and this could be used for DoS attacks. However, all nodes can check if the transaction is valid before forwarding it.

Blazej, Nicolas, Olivier

Here's how to replace linkable ring signatures by a ZKP. Let us define:

1. x

: a random number;

1. h

: Hash(x);

1. h_!

: Hash(!x);

1. L

: a list of h;

1. L_!

: a list of h_!

1. pk_v

: new public key / validator public key

1. \Delta_t

: A length of time, typically 3 months, less than the minimal duration a of a validator appointment.

- ValidatorContract#burnEth method

inputs: h

state: L

actions:

(1) burns 32 eth

(2) add h

to L

(3) remove from L

all h

older than \Delta_t

- ZK circuit

public inputs: L'

, h_!

, pk_v

private inputs: x

, h

, \mathit{pos}

: position of h

in the L'

vector

statements:

(1) Hash(x

) = h

(2) L'[\mathit{pos}]

= h

(3) Hash(!x

) = h_!

- ValidatorContract#registerValidator method

inputs: L'

, h_!

, pk_v

, ZKP(L'

, h_!

, \mathit{pk}_v

)

state: L_!

statements:

(1) ZKP is a valid proof

(2) all elements of L'

are in L

(3) L_!

does not include h_!

actions:

(1) add h_!

to L_!

(2) all the standard job of adding a validator

(3) remove from L_!

all h_!

which older than \Delta_t

.

And the scenario for the wannabe validator becomes:

a) U (1) chooses a random number x

; (2) calculates h

= Hash(x

)

(3) choose the pk_v

to use as a validator

b) U calls \mathit{ValidatorContract#burnEth(h)}

c) U waits for a while

d) U (1) select the h

from \mathit{ValidatorContract#L}

to be included in L'

. Calculate \mathit{pos}

(2) generates \mathit{zkp}

= ZKP(L'

, h_!

, pk_v

, x

, h

, \mathit{pos}

)

e) U calls \mathit{ValidatorContract#registerValidator(L', pk_v, zkp)}