co-authored with @0DanielTehrani

EDIT: added signature check on contentData

on recommendation from @enricobottazzi

In this post, we present a simple scheme for persistent (pseudo)nyms based on Eth set membership.

# Why?

Use of Ethereum naturally places one in a number of important, identity-defining ‘sets’. Sets like:

- all holders of NFT X
- voters in DAO Y
- liquidity providers to protocol Z at time T

We believe one’s existence within such a set is an identifier of growing importance in the crypto+internet economy. And we believe that being able to do so while maintaining privacy is a fundamental human right.

So why pseudonyms? We’ve observed (i.e. in heyanoun and heyanon) that while Ethereum users have things to say pseudonymously, without a persistent name to which they can accrue reputation, the flavor of interactions is quite simplistic.

This document lays out an extremely simple SNARK-based mechanism for persistent pseudonyms that allow reputation accrual.

# Properties

There are a few properties we designed this scheme around:

### nyms are ‘single-signer’

A given pseudonym must be attributable to no more than one Ethereum signer. Stated differently, a human viewing a given pseudonym’s usage anywhere on the internet must know that it’s being used by one Ethereum pub/privkey pair.

### linkage to Eth-based set membership

This one’s obvious, but worth restating. We want our pseudonyms to mean something relative to sets on Ethereum. As such, our proof mechanism must allow a pseudonym-holder to associate Eth-based sets they’re into pseudonyms they control.

### web3 wallet compatible

It’s crucial that our mechanism work with the ecosystem of tools/infrastructure that exists today. We have ideas for improvements that require heavier-weight changes to web3 wallets (An ECDSA Nullifier Scheme for Unique Pseudonymity within Zero Knowledge Proofs), but testing applications with these improvements would require many entities in industry (wallets) to make a uniform change to their infra.

### non-interactive, stateless

At this stage, we’re most interested in the simplest, easiest to use mechanisms. Too many layers of interaction could make something unusable and complicated economics could get in the way of agile experimentation.

Thus, we designed our mechanism to be stateless (i.e. no chain state management) and require minimal interaction from a user.

### (semi-)human choosable and (semi-)human recognizable

We want humans to:

1. be able to express themselves with their nym
2. be able to recognize/remember nyms they see online

2 is crucial for reputation accrual. We want our nyms to make sense to human minds.

1 allows nyms to be open to human creativity.

Because we're requiring statelessness at this point, we need to combine the 'human-grokkable' part of a nym with a collision-resistant ID, to avoid name collisions.

The result is nyms of the format {human-readable-segment}-{UUID}, i.e. 'alice-2443212411'

One way in which state might be added to nym ownership in the future is via an ENS-like registry of 'vanity' nyms, but we consider this outside of the scope of this note.

## Proof scheme

We utilize the following circuit pseudocode to associate nyms with activity:

circuit ProveNymOwnership { // hash is machine-readable UUID, code is human-readable segment pub input nymHash pub input nymCode

// ECDSASign(nymCode) priv input signedCode

// i.e. merkle root/proof pub input ethSetAccumulator priv input ethSetProof priv input ethPubkey

// content that user is producing from their nym pub input contentData priv input signedContentData

// check 1 signedCode === ECDSASignVerify(ethPubkey, nymCode) nymHash === Hash(signedCode)

// check 2 verifySetMembership(ethSetProof, ethSetAccumulator, ethPubkey)

// check 3 signedContentData === ECDSASignVerify(ethPubkey, contentData) }

check 1 ensure that the human readable and unique/UUID pieces of the pseudonym are related by a private signature. because the signature is private, the user's identity is concealed and because the UUID piece of the pseudonym is derived from the signature, we can be assured that only a single ETH signer can produce a given nymHash

check 2 is the same set membership check we've used in previous applications.

check 3 assures that the content is 'labeled' with this pseudonym, similar to use in previous applications.

The combination of checks here ensures that:

1. a given pseudonym is known to be associated with 1 or more Eth sets
2. only a single Ethereum signer can produce a given pseudonym
3. 2 signers are extremely unlikely to collide on nymHash

even if they choose the same (human-identifiable)nymCode

The nyms as shared on the internet (i.e. human-readable venues) can be of the form:${nymCode}-${nymHash}

, i.e. 'alice-2443212411' as mentioned before.

nymHash

can be be used in any venue where machine-readability is all that matters (i.e. interpretability in the EVM/smart contracts).

Finally, we use 1 or more unconstrained public inputs to pin this proof to the 'content' the user is attaching the proof to.

At first, this may simply be threaded messages (as in heyanoun v2), but we're keeping the core mechanism general enough for this content to be anything from a social media post to an Ethereum transaction. We believe this scheme can be used to label almost any piece of information on the internet with an Eth-based pseudonym.

## UX

To use a nym = ${nymCode}-${nymHash}

, a user creates a proof for the desired combination and the set they're claiming membership in. He/she also associates the proof to the piece of content he/she is creating via the unconstrained input contentData

Note that a user can't 'spoof' a known nymHash

; they can only produce nymHash

-s corresponding to ECDSA signers they control.

Users choose how to associate {nym, proof, anonymity set, content} online.

One slight hitch is that a user must keep track of nymCodes

they correspond to. Given that humans already remember social media handles, this doesn't seem like too much trouble.

However, even if he/she doesn't, as long as he/she has access to some content/nyms he/she might

have used in the past, the nyms can be tested by re-signing and seeing which nymHash

es correspond to his/her signer.

## Extensions

There are a number of extensions on this model that we're excited to flesh out and experiment with.

(briefly) 2 ideas:

1. use 'previous proof hash' in contentMetadata

as a way to create 'discussion threading'

1. 'nym/reputation' merging by generating a proof showing that one can create 2 or more nymHash

es

For brevity's sake, we'll save fleshing these (and other ideas) out for a future post!