Suppose that you want to generate a secret s

, where s

can be recovered by bringing together M

of N

secret-shares, where all N

secret shares are pre-known

. Two use cases of this are:

- A brainwallet where the N

shares are answers to N

security questions, and you want the funds to be recoverable with only answers to M

security questions (security questions suck individually

, but if you combine eg. 20 of them you can get quite a lot of entropy)

- A social recovery design where you want to use threshold decryption instead of smart contract wallets because you are trying to recover access to private data

and not cryptocurrency, and you want your recovery partners to be able to use keys that they already have (to reduce risk that they will lose those keys)

Plain old M-of-N secret sharing does not work for either of these use cases, because it only allows M

shares to be pre-chosen; the remaining N-M

shares must be generated from the original M

using a deterministic algorithm, and look like random data (in the brainwallet case, making them unsuitable as answers to security questions, and in the social recovery case, requiring users to use special software to store them, instead of making them derived from an existing HD wallet).

So here is what we do instead. We make a N-of-(2N-M)

threshold scheme, generating the N-M

excess shares from the original N

. We then publish all N-M

excess shares on the blockchain

. If desired, in the social recovery case one could instead simply give every participant a copy of all excess shares. This has the effect that the excess shares become effectively public information: there is negligible risk that they will get lost, but also any attacker will have them. As a result, only M

of the non-published N

shares are needed to combine with the N-M

excess shares and uncover the data - hence, we have an M-of-N scheme, which is exactly what we want.

Edit 2021.07.18: alternative mechanism for the social recovery use case

In the social recovery use case, we want to make the setup procedure as simple as possible, because users are lazy and if setup is difficult they will inevitably choose insecurely small recovery partner sets. This means distributed key generation (DKG) needed to generate secret shares in a decentralized way is likely a bad idea, because it requires 2 rounds of communication (which implies either extra blockchain transactions or everyone being online at the same time and

having a synchronous communication channel).

Instead, we can take advantage of the fact that the account holder themselves has their private key. They can simply ask each recovery partner for their public key (eg. via pk = G * hash(ecdsa_sign(msk, nonce))

where msk

is the recovery partner’s main secret key), and then publish a single transaction on-chain containing nonce

and encrypt(share_i, pk_i)

for each i

(where share_i

is the i’th share of the key and pk_i

is the public key of the i’th participant).

If we are careful about not reusing nonces and thus not reusing keys (eg. setting nonce = hash(secret, maddr_1 ... maddr_n)

,where secret

is the value being put into recovery and maddr_i

is the address of the i’th recovery partner, should be sufficient), a very simple bare-bones Diffie-Hellman encryption can be used. This means that only a single transaction, with 32 * (n+1)

bytes of calldata, is sufficient to save the recovery info.