Hi, we've been recently doing some research on distributed randomness, want to share a RandHound-influenced protocol that has the properties from the title, would appreciate feedback.

(EDIT: the description below is fully rewritten based on some offline feedback)

(EDIT2: more formal latexified version can be found here:https://www.overleaf.com/read/pcrtmwpxvnkb)

n

participants to do the following:

1. Each participant j

generates vector r[j]

of size k = n*2/3

where each element is a 256 bit random numbers, erasure codes them to have a vector s[j]

of size n

with n

shares such that any k

shares can reconstruct the chosen k

numbers, and encodes each of the n

shares with the public key of one of the partipants to get a vector es[j]

of size n

.

They then publish es[j]

. Here it's important that nobody can recover r[j]

by just observing es[j]

1. Participants reach a consensus on a set S

of at least k

published es

's.

1. Each participant i

publishes decoded row of es[{S}][i]

. Once k

participants published the rows, everybody can reconstruct the r[{S}]

.

I now want participants for each j

for which r[j]

was sucessfully reconstructed to be able to reproduce the es[j]

and confirm that it matches the published es[j]

. If it matches, then all the participants were able to reconstruct r[j]

, no matter what shares they observed. If a participant failed to reconstruct the erasure code or it didn't match the es[j]

, then all the paritcipants either failed to reconstruct it or reconstructed something that doesn't match es[j]

.

1. Let S

’ be the subset of S

for which the r

was reconstructed. The output of the randomness beacon is the some function of the r[{S'}]

Here there are some requirements to the public key ecnryption:

1. Encryption needs to be determenistic, so that reconstructed es

in step 3 matches the published es

in step 1.

1. If some es[j][i]

is gibberish (i.e. doesn’t decrypt or decrypts into something that is not equal to es[j][i]

after re-encrypting), it should be possible to prove it.

Seems like ElGamal in which the step y=random()

is replaced with y=hash(input)

works for (1) above, and Chaum-Pedersen proof works for (2) if a malicious actor still used some y

that was not equal to y=hash(input)

.

Comparison to other schemes:

- This approach is naturally inferior to RANDAO+VDFs in that it has worse liveness and safety requirements, but VDFs have the known issues with the necessity to build ASICs (or fear that someone else will).
- It appears to be as good as threshold signatures (except that it requires n^2

network instead of n

for threshold signatures IIRC) without requiring the expensive DKG step.

- Compared to RANDAO it is unbiasable
- Compared to RandShare it has lower complexity, compared to RandHound it is significantly simpler.

Feedback is appreciated.