This is a fun subject, I wanted to sketch a rough idea that could probably be massively improved.

It is a slightly different threat model (in terms of a “trusted” or “untrusted” servers) than the proposed ones. It also requires very heavy computation for the servers (quadratic in the number of notes) but is lightweight for the client. It is adapted from these voting protocols: JCJ *and Selections*

Consider a set of 10 servers (trustees). The user is asked to trust that at least 1 is honest. The trustees generate a shared 10-out-of-10 public key for a threshold encryption scheme that is partially homomorphic. One candidate is Elgamal where public keys are the same format as signature schemes like ECDSA and Schnorr.

A big simplifying assumption (that I don’t know works in practice) is that the set of note recipients are known at the beginning of an epoch and if a user is not in this set, they cannot receive notes until the next epoch.

For each epoch, the protocol is as follows:

- Each recipient pre-registers to receive notes by choosing a pseudonym (as a random value) and encrypting the nym under the trustees’ public key. The encrypted nym is initially associated with the user it belongs to, as identified by their public key or address or any unique ID. So Alice with address Addr_A

posts \langle Addr_A, \mathsf{Enc}(Nym_A)\rangle

. The protocol must ensure this value is actually from Addr_A

either through signatures or by being sent in a transactions from Addr_A

.

- Assume 4 users register as follows:

Addr_A

\mathsf{Enc}(Nym_A)

Addr_B

\mathsf{Enc}(Nym_B)

Addr_C

\mathsf{Enc}(Nym_C)

Addr_D

\mathsf{Enc}(Nym_D)

- The trustees each take a turn performing the following tasks
- The first trustee takes the list and encrypts each element in the first column (in subsequent steps, they will instead rerandomize each already-encrypted values) and re-randomizes each value in the second column.
- The trustee then shuffles the list row-wise
- The trustee publishes the rerandomized and shuffled list
- The trustee produces a ZKP that the output list is a permutation of the input list (many proof schemes exist, including the Neff Shuffle* without revealing the permutation they used or the random factors they used
- The first trustee takes the list and encrypts each element in the first column (in subsequent steps, they will instead rerandomize each already-encrypted values) and re-randomizes each value in the second column.
- The trustee then shuffles the list row-wise
- The trustee publishes the rerandomized and shuffled list
- The trustee produces a ZKP that the output list is a permutation of the input list (many proof schemes exist, including the Neff Shuffle* without revealing the permutation they used or the random factors they used
- After the last trustee has shuffled the list, each trustees will check each ZKP and sign the list if it is valid. Once all trustees have signed the list, the trustees will employ threshold decryption to decrypt the Nym

column of the list. The result is called the \mathsf{Roster}

and it is valid for one epoch and it might look as follows:

\mathsf{Enc}(Addr_A)

Nym_A

\mathsf{Enc}(Addr_C)

Nym_C

\mathsf{Enc}(Addr_D)

Nym_D

\mathsf{Enc}(Addr_B)

Nym_B

- Note: as long as one trustee is honest, no one will know which user ends up in which row of the table (except the user themselves who can identify their nym).

To send a note from Alice to Bob, Alice encrypts Bob's identifier \mathsf{Enc}(Addr_B)

and includes this in the payload of the encrypted note. Call \mathsf{Enc}(Addr_B)

the \mathsf{Hint}

.

- Note: Because our simple notation does not include random factors in \mathsf{Enc}

, it is important to note that this value \mathsf{Enc}(Addr_B)

will not be bitwise identical to the corresponding value \mathsf{Enc}(Addr_B)

in the 4th row of the \mathsf{Roster}

. Both represent an encryption under the same public key of the same message but will have different random factors and thus be different ciphertext values. In fact, for Elgamal, they will be computationally indistinguishable from an encryption of a different message (under standard cryptographic assumptions).

- Trustees will take the \mathsf{Hint}

and conduct a plaintext equality test (PET)* between each encrypted key in the first column of the \mathsf{Roster}

:

\mathsf{PET}[\mathsf{Enc}(Addr_A),\mathsf{Hint}]\rightarrow False

Nym_A

\mathsf{PET}[\mathsf{Enc}(Addr_C),\mathsf{Hint}]\rightarrow False

Nym_C

\mathsf{PET}[\mathsf{Enc}(Addr_D),\mathsf{Hint}]\rightarrow False

Nym_D

\mathsf{PET}[\mathsf{Enc}(Addr_B),\mathsf{Hint}]\rightarrow True

Nym_B

- Note: a PET is very close to a threshold decryption in cost, it just requires a pre-processing step before decryption. So doing a PET with every value in the table is basically trial decryption, it is just being done by the trustees instead of the user.

- The trustees will then add Nym_B

to the encrypted note.

- Bob will look through all notes for anything marked Nym_B

.

Additional notes and optimizations:

- The protocol is not perfectly private as notes received by the same recipient in an epoch can be linked together.
- Users can reuse Nym_A

but this extends the linkage possible.

- Users can supply a batch of \langle\mathsf{Enc}(Nym_{A,E1}),\mathsf{Enc}(Nym_{A,E2}),\ldots\rangle)

values for a set of future epochs E1,E2,E3,\ldots

at the start of the protocol.

- The threshold decryption can be m-out-of-n for different values of m and n. For example, if it were 7-of-10, then the protocol could proceed when 3 or less trustees were offline, however users would have to trust that at least 4 trustees are honest rather than 1.

I had to remove links to papers for the concepts marked with * because I am a new user on this forum and it tries to prevent spam.