Single key per address. In practice, people may want to store multiple public keys associated with their address. The simplified protocol gives no mechanism to do so. As a solution to this, we can allow the key to be the hash of an address concatenated with an optional arbitrary string. This allows, for example, Bob to store an application key at `hash(‘‘4155551212 || application_name’’)`, or an ephemeral application key at `hash(‘‘4155551212 || application_name || 20171117’’)`.

Node failure. Any model that relies on a single node to maintain state is susceptible to that node failing. We can address this by having multiple nodes participate in maintaining the state. (In doing so, we must also ensure that only a small number of nodes send a secret message to a user issuing a attestation request, to avoid overloading the user.) In this model, the secret message must also be verifiable by other validators, even if they did not construct it. This is achieved by signing the message with the private key of the validator sending it. To avoid repeat-attacks, each message from the same validator must be unique.

Malicious Validator. A malicious validator node may choose to bypass the message/response step, and instead, write an entry to the ledger in which they choose somebody else’s address, generate their own key pair for that address, and then sign the secret message with the private key that they generated. Doing so allows the validator to spoof an address, claiming payments intended for somebody else. We can address this by requiring consensus between multiple validators who have no mechanism to collude.

Transaction Transparency. If we are using hashed phone numbers as public keys, then a traditional bitcoin-style blockchain will allow a user to see the transactions of the contacts in their address book. We can address this by implementing the computationally efficient version of zk-snarks as described in [12].

DDoS. Finally, a malicious user may submit thousands of bogus requests to the validator, both tying up the validator and effectively using the validator as a spam agent. We can mitigate this by introducing a cost to attestation.

2.1.3 Distributed Scheme

We introduce here a distributed scheme that introduces each of the features suggested above. In this scheme, rather than the single validator node we describe in Section 2.1.1, a peer-to-peer network of multiple validator nodes maintains the database. The network is open and permissionless; anybody may join as a validator, and validators may leave and rejoin the network at will. Each validator maintains a full copy of the attestation pending queue and attested user database. For each attestation request, validators are chosen at random to handle the attestation.

An attestation workflow would then look like this. First, a user will issue an attestation request by sending the request to the Attestations smart contract, along with an attestation fee. The Attestations contract then selects a validator at random from the validator set and generates a message for the validator; the validator then signs that message, sends it to the registrant, who also signs it and sends it back to the Attestations contract. The Attestations contract then verifies the signatures of the registrant and the validator, and if they match, then records the attestation. Most dapps will require multiple attestations, in which case, if there are not enough attestations recorded on the chain, they will simply request more.

Having multiple validators addresses the node failure issue. Requiring multiple attestations addresses the malicious validator issue. The attestation fee addresses the DDos issue. And the attestation requests are issued as a hash of the `(address | pepper | application string)`, so as to avoid address harvesting, and to allow for multiple keys per address.

2.1.4 Summary of Operations

An alternative way of framing the protocol is in describing the roles and operations allowed to each node in the system.