Algorand Governance:
Smart contract(s) for claiming rewards

Each governance period will have an escrow account for holding the rewards pool for that period. At the conclusion of each governance period, the Algorand Foundation will sign a list of all the compliant governors' accounts, namely those who maintained their committed Algo level throughout the period and participated in all the votes. The reward rate for governors in that period will be calculated as the ratio between the reward pool for the period and the number of Algos committed by compliant governors' accounts, rounded down to microAlgos. Any amount left in the escrow account (if any) will be refunded to the main rewards pool account.

The signed list will be used to claim the rewards of compliant governors' accounts, by making a smart-contract call to the escrow account that holds the rewards pool. For the benefit of governors whose wallets do not support smart-contract calls, we allow any arbitrary Algorand account to initiate that call, not just the governor's account to which the rewards should be sent.

The rewards-claiming application will consist of a stateless TEAL that controls the escrow account ES (and also verifies the required signatures), and a stateful application AP that keeps track of who already claimed their rewards. The stateful application therefore needs to keep one bit per governor, which is implemented as a bit-array. This bit-array could in principle span the local state of multiple accounts, but in practice we expect it to fit in the local storage of just one account.

In more detail, each signed entry in the list that the foundation prepares will have the format `entry = (period, state_keeping_addr, bit_index, governor_addr, amt)` of type `(uint16, address, uint16, addr, uint64)`, encoded using the ABI conventions in https://github.com/algorandfoundation/ARCs/pull/7, where:

- `period` is the governance period in question, a small integer (1,2,3,...);
- `state_keeping_addr` is the Algorand address where the bit of state for that governor is kept;
- `bit_index` is the index of the bit for that governor inside the local state at the state-keeping-address;
- `governor_addr` is the Algorand address to which the rewards should be sent;
- `amt` is the rewards amount that this governance address has earned, in MicroAlgos

Each entry like that will also have a signature field, signing the above tuple relative to some Algorand Foundation keys that are used for that governance period. We will have some fixed number N of keys, and an entry will have to be signed by at least T of these N keys to be considered valid. (We will use N=3 and T=2.)
The Foundation public keys that are used to verify the signatures, as well the period number, will be *hard-coded in the stateless program controlling the escrow account ES*. The signatures

must be verifiable by the TEAL code of ES (using the opcode `ed25519verify`). Concretely, the following byte string will be signed:
`"ProgData" || ESCROW_program_hash || enc(entry)`
Where:

- `||` is the concatenation operator
- `ESCROW_program_hash` is the 32-byte address of ES without the checksum
- `enc(entry)` is the encoding of entry following the ABI conventions

The Algorand Foundation will also prepare as many state-keeping addresses as needed to keep track of all the governors for the current period, and will opt-in each one of these state-keeping addresses to the governance application. In addition, it will store in each of them the escrow account address (which implicitly ties them to the governance period and the foundation's public key). The Foundation will then re-key these state-keeping accounts to an address with no secret key, thereby preventing them from doing anything else beyond serving as state for the governance application. (In particular it will prevent them from issuing a clear-state call.)

To claim the rewards for a particular governance account (call it GA), an arbitrary calling address (denoted CA) will issue a two-txn group as follows:

1. A payment transaction from ES to `governor_addr` with transaction fee equals twice `MinTxnFee` and the amount equals to `amt - 2*MinTxnFee`.

   Note that in case of congestion, it may take more than `2*MinTxnFee` to get the transaction group accepted. In that case, it is up to the CA to provide the missing fees with the call to AP.

2. An application call to AP with the following method:
```
{
  "name": "claim",
  "desc": "Claim the governance rewards for governor_addr
account",
  "args": [
    { "name": "bit_index", "type": "uint16" }
  ]
}
```
   with the foreign account `state_keeping_addr`.

The stateful governance program

The creator of the stateful program AP will be an admin account whose key is held by the Foundation. Once deployed, this application will only accept updates from its creator (which will be a 2-of-3 multisig account to enhance security).

The application will support two types of calls: one used by the state-keeping addresses to initialize the state (before they re-key to zero), and the other used by governors to claim their rewards.

State-initializing calls

Any account can become a state keeping account by making an opt-in application call with the following method:

```
{
  "name": "optIn",
  "desc": "Initialize a state-keeping address with an all-0 array",
  "args": [
    { "name": "escrow_addr", "type": "addr" }
  ]
}
```

This call will opt-in the caller and store escrow_addr in its local storage, in addition to a size-M bit array, all bits initialized to 0. This call will fail if the caller is already opt-in.

M is a parameter that is set when the application is created, we use M=15x127x8=15240: One key-value pair will store the escrow address, leaving 15 key-value pairs for the bit array, and each one will have a 1-byte key (from 0x00 to 0x0e) and a 127-byte value.

The foundation will use the above call to initialize the state-keeping addresses, and will also make these addresses non-participating and re-key them to a dummy address with no secret key, preventing any further actions from them.

Note that anyone can add state-keeping addresses like that, but only the ones that are included in the list that the foundation signs (using the keys that are hard-wired in ES) will be usable for claiming rewards from ES.

Rewards-claiming calls

Any account can make a rewards-claiming application call to AP, using the `"claim"` method above with foreign account `SA`. The application program AP will then check the following conditions:

- It is the second transaction in a group of size two
- The other transaction is a pay transaction
- The escrow address in the local state of SA is the same as the sender of the pay txn
- The bit at position `bit_index` in the local storage of SA is set to zero
- Sanity checks (specifically onCompletion=NoOp)

If all these tests pass, then it sets the bit at position `bit_index` in the local storage of SA to one, and approves its transaction.

Note that the only thing ensured above, is that change in state will happen if and only if the escrow address stored at SA approves the corresponding pay transaction. For the SA addresses that the Foundation prepared, that escrow account will be the one controlled by the governance stateless-TEAL program, which has the Foundation public keys and period number hard-coded, and is making many checks (see below).

The stateless governance program

The escrow account is controlled by a stateless TEAL that has hard-coded the application ID of AP, N public keys `pk_1,...,pk_N`, and a period number `prd`. When used to authorize a transaction, it expects N arguments `sig_1,...,sig_N`, of type `byte[]`, and verifies the following:

- The transaction is a payment
- The next transaction in the same group is an application call to AP, using the method `"claim"`
- At least T of the N arguments have `len(sig_i)= 64`,
- For `i=1,2,...,N`, if `len(sig_i)= 64` then it is a valid signature wrt the hard-wired key `pk_i` on the string

  `"ProgData"||ESCROW_program_hash||enc(entry)`

  where `entry = (prd,SA,bit_index,GA,amt)` of type `(uint16, address, uint16, addr, uint64)`, with
  - `prd` is the hard-wired period number
  - `amt` is equal to the sum of the payment amount and the fee
  - `GA` is the receiver of the payment
  - `SA`, `bit_index` come from the application call to AP

  and `enc()` follows the ABI encoding rules.
- The transaction fee is at most twice `MinTxnFee`
- Sanity checks (no reKey, no closeTo)

Note that rewards can be claimed as soon as the signed list is published, the above does not check anything about `firstValid`.

In addition, we want the escrow account to be non-participating, so the smart-contract will also accept a transaction to make it so. The account will have hard-wired first/lastValid rounds for that `makenonparticipating` transaction, and we will use a lease to ensure that only one of these can be approved.

Application Timeline

The stateful application will be generated and deployed first, before the stateless code is finalized, note that this program does not depend on the address of the escrow account. Once the application is deployed, we do the following for each governing period:

1. Choose a set of new key pairs and store the secret keys safely, denote the public keys by `pk_1,...,pk_N`.
2. Finalize the stateless program with `pk_1,...pk_N` and the period number `prd`, and derive the escrow account address ES.
3. Fund the escrow account ES from the rewards pool, make it non-participating, and open the governor sign-up for this period
4. Once the sign-up window is over, we will determine the number of state-keeping accounts that's needed (most likely only one). For each one of them, generate a new account and issue the following transaction:
    - A pay transaction to fund this new account with the required minimum balance
    - Opt-in from the new account to the governance application, calling the `"optIn"` method to store ES
    - Make this address non-participating
    - Re-key this address to a dummy key that has no secret key
5. When the governing period is over, generate the list of compliant accounts, and for each one sign a tuple as above with T of the N keys.

Optionally, the Foundation can also set up a smart-contract account for making the application calls for reward claiming. That program will be willing to approve any application call, as long as it does not need to pay the fees (and the transaction does not rekey). Funding such an account with the required minimum balance will make it possible to generate the transaction group for claiming rewards without having to sign anything (except in cases of congestion, where the application call needs to pay fees).

Pseudocode Implementation

The stateful application

```
main(args)
{
  if this is a new application-creation call, accept
  else if txn.onCompletion==update and txn.sender==creator, accept
  else if txn.onCompletion==optin
    call optIn(args)
  else if txn.onCompletion==noop and args[0]==claim-selector
    call claim(args)
  else reject
}

optIn(escrow_addr=args[1] of type addr) -> void
{
  localStorage['ES'] = escrow_addr
  for i=0 to 14
    idx = last byte of itob(i) # this is the low byte of i
    localStorage[idx] = Byte(0)-repeated-127-times
  accept
}

claim(bit_index=args[1] of type uint16) -> void
{
  prev = txn.GroupIndex -1 # panic if less than zero
  if gtxn[prev].type!=pay, reject
  if bit_index > 15*127*8, reject

  sa = foreignAddr[1] # panic if not provided
  if gtxn[prev].sender != storage[sa]['ES'], reject

  idx1 = itob(bit_index / (127*8))
  if len(storage[sa][idx1])!=127 bytes, reject

  idx2 = bit_index mod (127*8)
  theBit = extract bit in position idx2 in storage[sa][idx1]
  if theBit!=0, reject

  store Bit(1) in bit position idx2 in storage[sa][idx1]
  accept
}
```

The stateless program

```
Hardwired constants: appID, claimMethodSelector,
     N, pk[0],...,pk[N-1], T, encodedPeriodNumber,
     nonPartLastValid, leaseVal

if txn.rekey!=null or txn.closeTo!=null or txn.fee>2*MinTxnFee,
     reject

if txn.type==keyreg and txn.lastValid==nonPartLastValid
   and txn.lease==leaseVal and txn.nonparticipation==true
     accept

next = txn.GroupIndex +1
if txn.type!=pay or groupSize<next+1, reject
if gtxn[next].type!=appl or gtxn[next].applicationID!=appID or
   gtxn[next].args[0]!=claimMethodSelector,
     reject

amt = itob(txn.amount + txn.fee)
ga = txn.receiver
sa = gtxn[next].foreignAddr[1]
idx = itob(gtxn[next].args[1])
entry = encodedPeriodNumber | sa | idx | ga | amt

verified = 0
for i=0 to N-1 {
  if len(args[i])==64, verified += ed25519verify(entry,args[i],pk[i])
}
accept if verified>=T, else reject
```