💡 DeNS v4: Explanatory paper 🧻

General logic

The system consists of multiple contracts, each of which carries a specific logic:

- **D4Root** – the primary router and coordinator that creates all other contracts and provides resolution services for them;
- **D4User** – a contract that corresponds to another internal wallet or external user by a pubkey (experimental feature!), a “wallet” in TIP3 terms;
- **D4Base** – the immutable contract that is deployed with specific static data to act as a basement for other contracts, therefore address of those other contracts is deterministic no matter their code, a “platform” in DeNS v1 / v2 terms;
- **D4Cert** – contains information about a specific certificate (“domain”);
- **D4Auct** – represents ongoing auction or reservation for a certificate name;

Storage layout

Necessary information is stored in those contracts, clarification of which follows:

- **D4Root** contains minimal systemwide information:
  - Images for all contracts (base, cert, auct, user) and their revision nums
  - Owner and pending owner block, also SMV root address
  - Configured auction disabling timeout and admin rights mask
- **D4User** contains useful and required information for a specific user:
  - Funds lock configuration (for auction returns);ssa
  - Arbitrary blackbox user masterkey (client-side encrypted);
  - Book of auctions, their information and placed bids (required by logic);
  - Book of mostly owned certificates for quick access;
- **D4Cert** contains owner block, value block and some time information,
- **D4Auct** contains:
  - Timing information about phases boundary;
  - Some utility and internal information (duration days or expiry base);
  - Number of bids placed and revealed
  - First and second bid runners and their investments

Some more useful information will follow

Generic flow logic

Terminology: call – off-chain (getter), invoke – on-chain (internal message), execute – on-chain (signed external message).

Resolve a domain some/domain/name **for entry type** index

1. Call D4Root.resolveFull(0, "some/domain/name"), get *certificate address*
2. Check existence of certificate at received address
3. Check expiry (D4Cert.expires) of certificate vs now
4. Call D4Cert.getValue(index), the call may fail if value is not set (not exists)

Alternatively, instead of D4Root.resolveFull, can do following instead of step 1:

1.1. Call D4Root.resolve(0, "some", *root address*) and get "some" cert address

1.2. Call D4Root.resolve(0, "domain", *"some" address*) and get "domain" cert addr

1.3. Call D4Root.resolve(0, "name", *"domain" address*) and get *certificate address*

With such approach it may be possible to check ownership, status and expiry of each domain in the chain instead of only the final one.

Begin working with user for a specific wallet

1. Check existence of deployed D4User: call D4Root.resolveUser(*wallet address*)
2. If user does not exist, invoke D4Root.deployUserForMe from user's wallet

   Then verify existence again and make sure the D4User exists.

3. Check D4User.masterKey.

   If it is empty, generate random 256-bit value (*master key*), ask user for the encryption password (maybe twice) and call D4User.setMasterKey(*encrypted master key*). Encryption shall guarantee integrity and validity of the used key, otherwise also need to store some MAC of *master key* such as hash of it.

   If it is not empty, ask user for the password and try to decrypt it for *master key*. If decryption fails (integrity check fails) ask for the password again. It may be possible to have some password reset mechanism (which will trigger the empty master key scenario) but with explicit warning of irreversible inability to reveal current bids.

   Derivation of internal encryption key and nonce is to be discussed.

Retrieving locked funds

If funds are returned from auction they are locked until reveal phase ends to prevent cross-user hopping and laddering. To retrieve them, you can

1. Call D4User.withdrawable() to determine available amount

2. Invoke D4User.withdraw specifying destination address (user wallet) and amount

Anyone can invoke D4User.sweepLocks to clean fundLocks of expired locks. It is implicitly invoked by the withdraw function to update totalLocked.

Administering certificate

To administer certificate the user has to invoke specific proxy function of D4User from the user wallet. These include:

setValue, resetValue, clearValues, certWithdrawExcess, certRequestUpgrade, requestProlong, deploySub, syncSub, certTransferOwner, certCancelTransferOwner, certAcceptTransfer, certRelinquishOwner

First (target) parameter is the address of the certificate, while remaining are of funct.

The certBook + certInfo should carry up-to-date information about your certificates, but under some conditions it may be stale, so should be verified on-chain.

Making a bid in auction, possibly creating it for some-domain

Auctions are currently only possible under root directly, not for sub-domains

1. Call D4Root.resolveAuction("some-domain"), verify existence of auction

   If the auction does not exist, resolve corresponding certificate

   If the certificate does not exist or is expiring (Sys.ExpiringAuctionPermit) invoke D4User.createAuction specifying domain name and desired duration in years

   Then verify existence again and make sure the D4Auct exists.

2. Verify that duration of the auction is desired by user (if not created) and that the auction is in correct phase (in bid phase, not yet in reveal phase).

3. Invoke D4User.makeBid providing auction address, encrypted *amount* with integrity guarantee (*data*) and *hash* (see D4User.utilBidHash for implementation example) of the bid. *nonce* should be derived from *master key* and be unique as per auction instance and user (using user address, auction address, and auction start time)

Revealing the bid

1. Check the known auctions to user (D4User.auctBook, auctInfo, auctBids), choose the one to reveal bid and re-verify information on-chain, check time phase.

2. Decrypt the encrypted *amount* and invoke D4User.revealBid providing the *amount*, derived *nonce*, and the auction address. The message shall carry at least *amount* + Sys.AddToReveal excl. fees (somewhat even more, extra will be returned)

3. It should be good to check the result on-chain, if the reveal was reflected

Finalizing the auction

1. Same step as in revealing the bid or user can specify any auction to finalize

2. Invoke D4User.finalize providing the address of auction to finalize

It is useful to finalize auctions where you win, but any user can finalize any auction if it is in correct phase.

Requesting prolongation of expiring auctioned certificate

Owner of expiring certificate can request prolong for (Sys.ProlongMultiplier / Sys.ProlongDivisor) * winning amount in auction for some amount of time after reveal phase ends (Sys.ProlongGrace). To do that, the owner can use requestProlong.

The user can request prolong even after that (D4Auct.finalizeAfter) but then anyone will be able to finalize auction at once preventing the prolong from carrying out.

Root administration

For testing, debugging, administration and initialization purposes, there are some functions in root: adminDeploy, adminReserve, adminChown, adminChexp, adminUpgradeUser, adminUpgradeCert that require corresponding admin_enabled flags to be set.

For production, admin can D4Root.dropAdminFlag them to disable that functionality permanently and irreversibly.

Ideally, administration should be passed to SMV, for that admin has to execute D4Root.assignSmvContract, then D4Root.dropAdminFlag(255) and finally D4Root.relinquishOwner two times to reset ownership to zero.