DC DARKPAPER

April 25, 2016

Intro

darkclam (DC) is a virtual machine sidechain, concept digital currency, digital identity system, private chat system, and p2p network. Andu (51) came up with the initial concept and distribution method. I perfected it and added my own ideas. I also give credit to creativecuriosity for helping me develop some of the virtual machine concepts. Any references to “me” or “I” in the paper are referring to the writer of this paper, daXXog. darkclam is shortened as DC throughout this paper. This paper is the result of months of work and research.

History

On January 27, 2016, I created the darkclam XCP asset with an initial distribution of 24,342,491 divisible tokens. 11,188,970.0521319 DC were sent proportional to wagered amount on the Canadian gambling site, just-dice.com to anyone who participated in the free drop process. I then announced a vote on March 02, 2016, that allowed the holders of the newly created asset to vote on the future distribution of DC. The voting process ended on March 16, 2016 and the remaining 13,153,520.94786813 DC were sent to a burn address, taking them out of circulation and making them unspendable. At the time I am writing this paper, a bug in counterwallet is currently preventing me from “locking” the asset, although I do plan on doing that as soon as the bug is resolved.

Future

darkclam will run as stand alone client that connects to the official CLAM client via RPC. On initial start, the DC client will index the transactions in the CLAM blockchain and order them by transaction number and block number. This simple dataset will be stored as an Array of transaction Objects in a database (aka “the stack”). When a new block is found, a blocknotify callback is sent to the DC client. The DC client then pushes confirmed transactions onto “the stack”. For each transaction pushed a function is called which parses the transaction data and passes it to the appropriate virtual machine. If a block is orphaned a “reverse” function is called which “undos” the stack pushes to keep everything in sync with the CLAM blockchain.

DC Virtual Machine

The darkclam virtual machine will be based off the Ethereum VM. It is a simple turing-complete machine that has an internal state that is modified by transactions on the CLAM blockchain. You can think of each CLAM transaction as a “command” to this virtual machine. The CLAMspeech field will contain the majority of the information involved in a darkclam transaction. Users can register their own virtual machines by paying a fee.

DC Addresses

darkclam addresses will be exactly the same as CLAM addresses. The originating address from the first input in a transaction is sent along with the CLAMspeech field and DC stack position to the appropriate virtual machines. This puts the transaction verification from the CLAM blockchain right into the DC system. Each command has a set “fee” in DC. This fee is deducted from the originating address and permanently removed from circulation. If the originating address doesn't have the enough DC, the command won't execute. At some point in time a “snapshot” will be taken on the darkclam XCP token. This snapshot will contain the information about the initial DC distribution. Users can then “dig up” by importing their private key from counterwallet.

Command Parsing Rules and CLAMspeech Syntax

alias [alias] [data]
This command creates or updates a human readable “alias” for a user. This command costs 1 DC to execute.

alias
Required. The name of your alias, formatted in lowercase letters, numbers, and dashes. The names “alias”, “vm”, “darkclam”, “lock”, and “transfer” are reserved. Aliases use the same “namespace” as VMs.

bitmessage
Optional. A bitmessage address to link to your account.

vm [asset name] [magnet link] [gas limit]
This command creates or updates a virtual machine that can be issued commands. This command costs 1 DC to execute plus any “gas” needed by the virtual machine to run.

asset name
Required. The name of your virtual machine asset, formatted in lowercase letters, numbers, and dashes. The names “vm”, “darkclam”, “lock”, and “transfer” are reserved.
magnet link
Optional. This link contains the initial state of the virtual machine. Subsequent vm commands can be issued to “force” a new virtual machine state or modify the rules of the virtual machine as long as they come from the owner of the virtual machine and the virtual machine hasn't been locked. A JSON file will also be included with the VM image which contains the abi, dependencies, version information, descriptions, and styling for the asset.

gas limit
Optional. The maximum amount of DC this command is allowed to consume. If unset, it goes by the default gas limit of 0.5 DC.

transfer [alias] [asset name] [destination]
This command transfers control of the asset or alias to a new owner. This command costs 0.5 DC to execute and it must originate from the current owner.

alias
Optional. If you set this to “alias” and the command has three arguments, this signals an alias transfer. Alias transfers do not change ownership of the underlying assets. Commands that use the alias after this transaction will route to the new owner.

asset name
Requried. The name of your virtual machine asset, formatted in lowercase letters, numbers, and dashes.
destination
Required. The DC address to transfer ownership to.

lock [asset name]
This command permanently locks the specified asset. No further “hard changes” to a virtual machine are allowed after this point.
asset name
Required. The name of your virtual machine asset, formatted in lowercase letters, numbers, and dashes.

[asset name] [gas limit] [command] [args]
A custom command to a virtual machine. This command costs a variable fee in DC based on the Ethereum gas system.

asset name
Requried. The name of your virtual machine asset, formatted in lowercase letters, numbers, and dashes.

gas limit
Required. The maximum amount of DC this command is allowed to consume. If unset, it goes by the default gas limit of 0.5 DC.

command
Required. The “entry point” function to call, defined in the ABI in the VM JSON file.

args
Optional. Additional arguments to pass to the virtual machine.

DC Virtual Machine Syntax
darkclam [amount] [[to 1], [to 2], [to 3], …] [gas limit]
This command sends a set amount of DC from the originating address to a destination account. The amount is split evenly between each destination. For multiple destinations the input amount must be divisible by the number of output destinations. This command costs whatever amount is needed to fulfill the gas requirements of the VM. This amount should be very small and can be approximated using an algorithm.

amount
Required. The input total amount.

to
The destination account(s). Can be a DC address, alias, or VM. VM's can use DC via Cross-VM communication.

gas limit
Optional. The maximum amount of DC this command is allowed to consume. If unset, it goes by the default gas limit of 0.5 DC.

Burning Stakes (PoBS)
The PoBS system is not actually needed to "prove" anything. It simply exists for the creation of new coins. Technically, you could start burning CLAM stakes today and be able to spend your DC when the full client is released.

First set up your CLAM client via the rpc console to burn staking rewards:
setrewardto xDARKCLAMxBURNxSTAKEXXXXXXXXXNaAaz

Then set it up to reward you with 1.0 darkclam per stake burned.
setspeech [your darkclam address here]

If a valid address was sent and the 1.0 CLAM reward was burned your address is rewarded with 1.0 darkclam.

VM Internals and Cross-VM communication

The darkclam hypervisor manages:

* storage/memory management
* checkpoints / history
* magnet system
* eXtended/abi/cmd/json/system xacjs
* gas management
* rpc system
* extended ABI system (new types)
  - Input -
    - Magnet - magnet links as data input
  - Output -
    - Function - a specially encoded function call, can be used to call code in other VMs
* dependencies
* Cross-VM communication

where new DCVM(**Buffer** *bytecode*, **Buffer** *trie*, **Object** *abi*)

(note, this constructor is usually called from the Hypervisor)

bytecode - The VM bytecode, loaded from the [package-name].vm file

trie - The previous VM state, a Patricia Merkle Tree expressed as a Buffer

abi - The abi definition as a JavaScript Object, loaded from the [package-name].json file

DCVM.run(**Integer** *index*, **Buffer** *origin*, **Array** *args*, **Function** *cb*(**Buffer** *trie*, **Error** *err*, **Buffer** *return*, **Integer** *gasUsed*, **Array** *logs*), **Integer** *value*)

index – the unique index of this transaction in "the stack"

origin – the "origin address" or virtual machine (proves ownership)

args[0] - The function name (entry point)

args[1 - ARG_LIMIT] - The arguments for the function (automatically encoded)

cb – Callback Function, used internally by hypervisor

trie - The next VM state (hypervisor manages confirmations with a run/rewind system backed by the checkpoint system), a Patricia Merkle Tree expressed as a Buffer; if err return lastState :)

err - Graceful failing for VMs

return - Buffer data to return to the Hypervisor (for Cross-VM Communication)

gasused - Amount of gas used running the selected function, passed back to Hypervisor for Cross-VM Communication

logs - An Array of logs spit out by the function code

value - The value transacted, used for DC transactions between virtual machines.

Chat System

The darkclam client will have a built in chat system based on bitmessage. The "alias database" will index aliases to bitmessage addresses to allow for easy, built in, private chat.

P2P Network and "lite clients"

A simple p2p network (possibly based on bitmessage) will allow for "lite clients" including smartphones to connect to to "full clients". Using a Public/Private key system, a user for example can scan a code using a smartphone that is displayed on a "full client" to allow for instant "remote control" of a full client. The full client will allow for revoking/granting permissions and access. As long as a "lite client" has a private key that the full client has "allowed", they can do anything that a "full client" can do.

Unfortunately at the moment I only have enough time for development to have darkclam be a “side hobby project”. If any developers are interested in building this system, go for it. DC is the future.