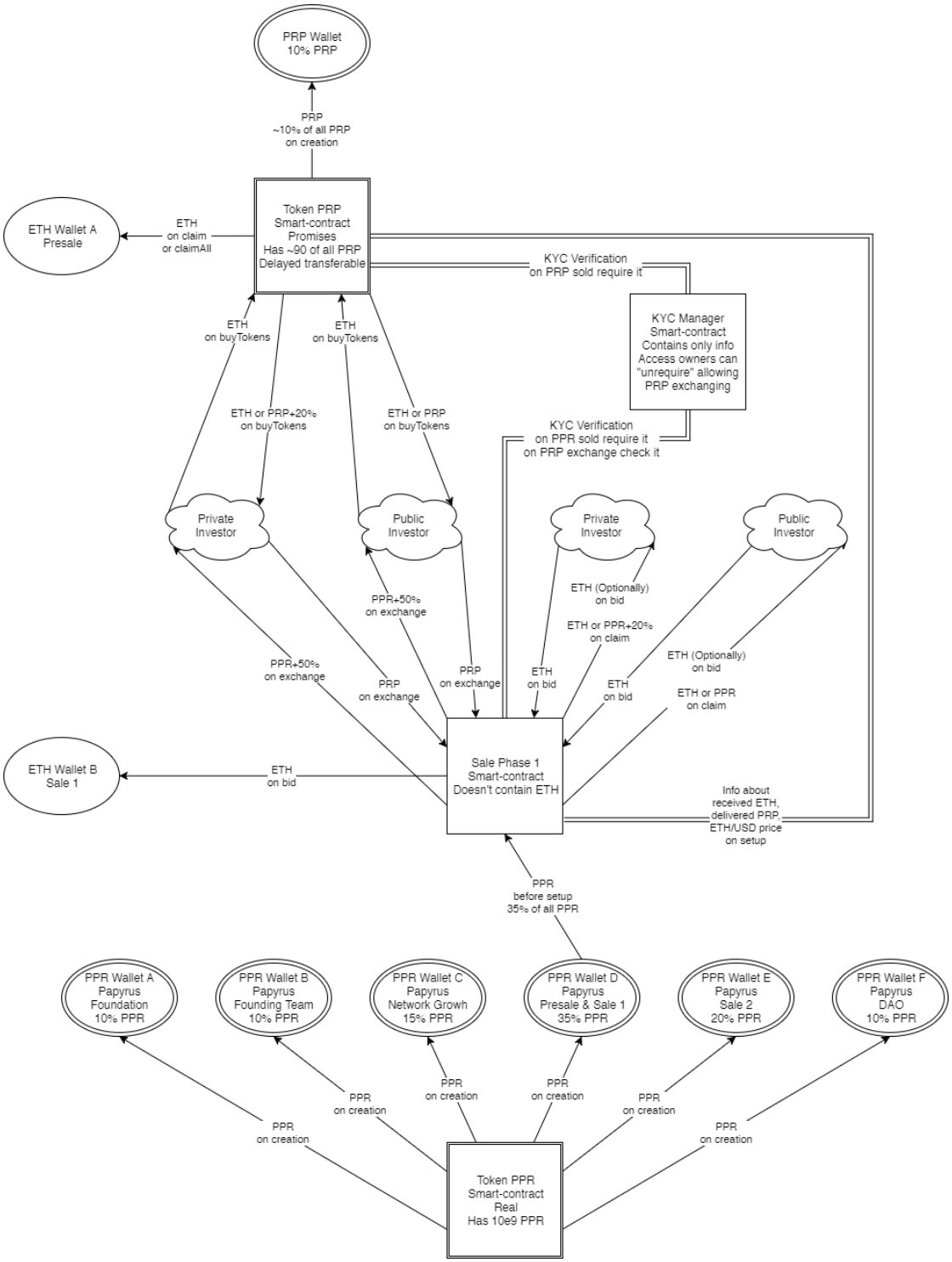
Introduction

Rough scheme of organization Papyrus Presale & Sale Phase 1 is presented below:



It is necessary to create Externally Owned Account (EOA) and protect its private key. This account will be used to create and manage core smart-contracts like Papyrus Token or Papyrus Presale. This is common practice to use single “owner” account to create and manage contracts. Let’s call this account **Core Account** further.

Multi-signature Wallet

There are several standard implementations of such common smart-contracts. For storing ETH received from crowdsales we will use one of these implementations. Most of them are pretty similar and have ability to limit amount of withdrawn ETH with some amount per day. To create such wallet it is necessary to specify list of owners, minimum required approves to perform transaction and amount of ETH allowed to transfer from wallet per day. I suggest to use 5 owners and minimum required approves as 3. Another possible configuration 2of3 instead of 3of5 (but it actually can be any). Day limit is also very useful option that prevents losing all stored ETH if something went wrong.

PrivateParticipation base smart-contract

PrivateParticipation.sol implements base smart-contract that just stores whitelist of addresses and amounts of weis allowed to invest. It has only function **allowPrivateParticipant()** with two arguments:

- _participant (address) - address of the participant of private stage of the auction
- _amount (uint256) - amount of weis allowed to bid for the participant

This function can be used only by owner of the smart-contract. When zero value as _amount is used it can means like denying private participant.

MultiAccess base smart-contract

MultiAccess.sol implements base smart-contract that stores whitelist of addresses and boolean values that are true if the address should have access to some actions on child smart-contracts. There is modificator accessGranted() that fails if address of sender has no access. It has only function **grantAccess()** with two arguments:

- _to (address) - address for which access should be changed
- _access (uint256) - address receives access when true and loses when false

This function can be used only by owner of the smart-contract.

Papyrus KYC

PapyrusKYC.sol implements ability store addresses that should be KYC verified. When PRP tokens are sold on auction all participants of the auction require KYC verification. Contract has two functions **isKycRequired()** and **setKycRequirement()**.

Function isKycRequired() can be used to determine if specified address should be verified. Returns false in cases when KYC verification is not required so for example PRP can be exchanged to PPR. It has only argument:

- _participant (address) - address of the participant

Function setKycRequirement() can be used to change requirement of KYC verification. It can be used only owner or from address that has access granted (MultiAccess.sol) and has two arguments:

- _participant (address) - address of the participant
- _required (bool) - new value for KYC requirement

Once function is called with argument _required as false it cannot be set to true then. Expecting changing values for addresses during all time is added as true -> changed to false forever.

Papyrus Token

PapyrusToken.sol file implements ERC20 token. It is inherited from OpenZeppelin smart-contract StandardToken. It has hardcoded name **"Papyrus Token"** and symbol **"PPR"** with **18 decimals** (just like ETH and many other tokens). This is not mintable token which means that there are pre-defined amount of **PPR** created – **1,000,000,000 (one billion) units**. Since these values are hardcoded and cannot be changed and tokens cannot be supplied, there will never be more than 1,000,000,000 **PPR** in the system.

During smart-contract creation from Core Account two arrays of addresses and amounts with the same lengths should be specified. There is hardcoded condition in smart-contract constructor which checks that **sum of all specified amounts should be equal to 1,000,000,000 (total amount of PPR tokens supplied)**. In else case smart-contract cannot be created. Thus we can specify array of 1 wallet address and 1 amount with value 1,000,000,000 to store all supplied **PPR** to single wallet, or we can divide this if we want to share some **PPR** with different set of owners (different multisig wallet).

PPR is transferable from the start. Users will be able to exchange and use in the Papyrus ecosystem within 30 days following the completion of **token sale phase 1**.

Pre-Papyrus Token

To perform token pre-sale we will create supplementary tokens for pre-sale participants, called **PRP** (pre-Papyrus). **PRP** tokens will be issued only once to conclude pre-sale and their amount will be limited to a fixed volume of **50,000,000 (fifty million) units**.

All **100%** of issued **PRP** tokens will be allocated for pre-sale process, including bounty and incentives for partners. **PRP** tokens will be transferrable in some time after pre-sale auction is finished. After **token sale phase 1** ends all **PRP** holders will receive main Papyrus **PPR** tokens in exchange to pre-Papyrus **PRP** tokens, after it all **PRP** tokens will be burned.

PrePapyrusToken.sol file implements ERC20 token. It is inherited from OpenZeppelin smart-contract StandardToken. It has hardcoded name **“Pre-Papyrus Token”** and symbol **“PRP”** with **18 decimals** (just like ETH and many other tokens). This is not mintable token which means that there are pre-defined amount of **PRP** created – **50,000,000 (fifty million) units**. Since these values are hardcoded and cannot be changed and tokens cannot be supplied, there will never be more than 50,000,000 **PRP** in the system.

During smart-contract creation from Core Account address of Papyrus KYC smart-contract and two arrays of addresses and amounts with the same lengths should be specified. Address of Papyrus KYC smart-contract is used to store all KYC verify requirements at some place where it can be used later by other Papyrus smart-contracts. List of addresses and amounts can be used to specify predefined list of wallets and amounts of PRP tokens to store. It can be useful for example to store 5-10% of PRP tokens on separate wallet so it can be used later for bounty program. bonuses or any else internal needs. All tokens that are not amounted in these two arrays are stored at PRP smart-contract itself. All these remaining tokens will be either sold or burned at the finish of auction.

Pre-Papyrus Token contract also implements simple crowdsale logic. Price of PRP token is constant during whole auction and specified before its start. Whose investors who participate in auction at private stage get 20% more PRP tokens for the same ETH. Auction stops when all tokens on contract balance are sold or when _auctionFinish block index is achieved. All unsold tokens are burned.

This contract should be able to handle bids done with ETH and BTC. We are still looking solution for security handling BTC transactions.

Smart-contract organized as simplest finite-state machine. Each state allows limited functions to be called. Here is full list of possible states:

- TokenDeployed
- AuctionReadyToStart
- AuctionStartedPrivate
- AuctionStartedPublic
- AuctionFinished

These states change linearly from up to down during auction.

Smart-contract creation

To create smart-contract you should already have created Papyrus KYC smart-contract and specify it’s address with 1st argument. 2nd and 3rd arguments are arrays of addresses and amounts described abode. After smart-contract is created its state becomes TokenDeployed.

Smart-contract creation

To create the Papyrus Presale smart-contract you should already have created multi-signature wallet that will be used to store received ETH. It also will be necessary to specify following arguments when creating smart-contract:_priceFactor (uint256) – auction start price factor that manages price curve [e.g. 4500]

- _auctionPeriod (uint256) – period of time when auction will be available after stop price is achieved in seconds [e.g. 604800 which means 7 days]

After smart-contract is created its state becomes AuctionDeployed.

TokenDeployed state

Only function that can be called at this state is **setupAuction()**. Only owner of smart-contract can call this function. You should specify following arguments:

- _wallet (address) – address of existing multi-signature wallet for storing ETH
- _ceiling (uint256) – auction ceiling in weis [e.g. 200000000000000000000000 which means 200k ETH and $50M if current price for 1 ETH is $250]
- _priceEther (uint256) – current price ETH/USD [e.g. 4000000000000000 if current price for 1 ETH is $250]
- _priceToken (uint256) - price of PRP token as PRP/ETH
- _bonusPercent (uint8) – percent of bonus tokens we share with private participants of the auction [e.g. 20]
- _minPrivateBid (uint256) – minimal amount of weis for private participants of the auction [e.g. 20000000000000000000 which means 200 ETH and $50,000 if current price for 1 ETH is $250]
- _minPublicBid (uint256) – minimal amount of weis for public participants of the auction [e.g. 2000000000000000000 which means 2 ETH and $500 if current price for 1 ETH is $250]
- _auctionPrivateStart (uint256) - index of block from which private auction should be started
- _auctionPublicStart (uint256) - index of block from which public auction should be started
- _auctionFinish (uint256) - index of block from which auction should be finished

Please, note that this function can be called several times before actual start of private auction. After it is performed first time state of smart-contract becomes AuctionReadyToStart.

AuctionReadyToStart state

You can call 5 functions at this state: **setupAuction()**, **setPrivateAuctionStart()**, **setPublicAuctionStart()**, **setAuctionFinish()**. Only owner can call all these functions.

Function setPrivateAuctionStart() can be called several times since it just update private auction start block index. It requires following argument:

- _blockIndex (uint256) – index of block from which private auction should be started

When investor calls function buyTokens() or just transfers ETH to the auction smart-contract, state of smart-contract can be changed to AuctionStartedPrivate if current block index equal or more than specified _blockIndex and current state is AuctionReadyToStart.

Function setPublicAuctionStart() can be called several times since it just update public auction start block index. It requires following argument:

- _blockIndex (uint256) – index of block from which public auction should be started

When investor calls function buyTokens() or just transfers ETH to the auction smart-contract, state of smart-contract can be changed to AuctionStartedPublic if current block index equal or more than specified _blockIndex and current state is AuctionStartedPrivate.

Function setAuctionFinish() can be called several times since it just update auction finish block index. It requires following argument:

- _blockIndex (uint256) – index of block from which auction should be finished

When investor calls function buyTokens() or just transfers ETH to the auction smart-contract, state of smart-contract can be changed to AuctionFinished if current block index equal or more than specified _blockIndex and current state is AuctionStartedPublic .

AuctionStartedPrivate state

It is possible to call 4 functions at this state: **buyTokens()**, **setPublicAuctionStart()**, **setAuctionFinish()**. Functions setPublicAuctionStart() and setAuctionFinish() can be called only by owner.

Function buyTokens() can be called by anyone but will work only for those callers whose addresses are allowed as private participants. It will be failed for all other callers. It requires following arguments:

- receiver (address) – address of previously registered private participant. This can has 0x0 value and in this case, address of caller will be used instead.
- customerId (uint128) – UUID v4 to track the successful payments on the server side

To make sure function will not fail caller should send amount of weis with this call at least _minPrivateBid specified previously.
Received ETH is stored on multi-signature wallet specified previously.
At success receiver gets PRP + bonus percent.

Functions setPublicAuctionStart() and setAuctionFinish() are described above and just can be used in several states.

AuctionStartedPublic state

Only 2 functions that can be called at these states is **buyTokens()** and **setAuctionFinish()**. Function setAuctionFinish() can be called only be owner.

During these states when function buyTokens() is called state of smart-contract can be changed to AuctionFinished. This will be happened if auction ceiling (_ceiling specified previously) is achieved or current block number becomes equal or greater than index specified in setAuctionFinish() previously.
Received ETH is stored on multi-signature wallet specified previously.
At success receiver gets PRP.

Function buyTokens() will fail at this state in case if caller address is registered as private participant.

Function setAuctionFinish() are described above and just can be used in several states.

ANY state

There are also several functions that can be called by anyone at any state of the auction:

- allowPrivateParticipant() – used to register private participants and can be called several times
- updateStage() – checks conditions and updates state of the auction if necessary, returns actual state
- burn() – burns (destroys) tokens from specified address with specified amount, can be used by owner or access granted address (see MultiAccess.sol)

After auction is finished it is necessary to call function setKycRequirement() on Papyrus KYC contract for each participant with 2nd argument false. This should be done only for participants who passed KYC verification. This will allow participant to exchange PRP on PPR later after Sale 1 is finished too.

Each pre-sale **PRP** token holder is granted to receive **PPR** tokens in exchange to his/her **PRP** tokens after token sale phase 1 ends using conversion ratio, determined as:

$$\mathbf{V_{PPR} = V_{PRP} * (P_{PRP} / P_{PPR}) * 150\%}$$

Where $\mathbf{V_{PPR}}$ – the amount of **PPR** tokens that **PRP** token holder will receive in a result of **PPR** token sale phase 1 in exchange to their **PRP** tokens, $\mathbf{V_{PRP}}$ – the amount of **PRP** tokens that token holder have, which are exchanged, $\mathbf{P_{PRP}}$ – price of **PRP** tokens denominated in **USD** and determined as a result of the **PRP** pre-sale auction in the day it ends (this price will be published by Papyrus team on the official website and in blog posts), $\mathbf{P_{PPR}}$ – price of **PPR** tokens denominated in **USD** and determined as a result of PPR token sale phase 1 auction in the day it ends (this price will be published by Papyrus team on the official website and in blog posts). We will determine tokens **USD** price using their **ETH** price determined just before start of each auction on one of the top crypto exchanges (we will publish details of this process on the official website and in blog posts).

Thus participants of pre-sale get strong incentive to participate as they eventually get 50% more **PPR** tokens than token sale phase 1 participants for the same **USD** volume.

For example, if anyone buys **PRP** for $10k at pre-sale, than he is guaranteed to receive PPR tokens for $15k at token sale phase 1 in exchange to his **PRP** tokens. Actual **USD** price of each token will be determined as a result of reverse Dutch auction in pre-sale for **PRP** tokens and in token sale phase 1 for **PPR** tokens accordingly.

Wider description of functions and other difference will be described later when we will get some working prototype coded.

Papyrus Sale Phase 1

Sale Phase 1 will be on of two big auctions. Auction is implemented as reverse Dutch auction, where everyone can participate. Auction will end when either of the following ending criterion is met:

- approximately **X USD** denominated in **ETH** worth of **PPR** is sold
- **35% of all PPR** tokens are sold
- pre-sale period of **14 days** ends

As a result of sale 1 auction PRP holders can change PRP to PPR using this contract. Also all participants of sale 1 can claim PPR in amount based on their bids and actual result of auction. Probably this should be required to support automatic claiming and exchanging all PPR w/o calling functions from participants.

This contract is in development yet so it does not contain a technical description for now.

Tokens Movement

Please look at the scheme presented above.

ETH:

- Investors send their ETH using function buyTokens() or bid() of one of two smart-contracts pre-sale or sale phase 1
- Received ETH during auction are transferred to multi-signature wallet attached to auction
- Since ETH Wallet A/B are multi-sig daily limit wallets ETH can be withdrawn for company needed under daily limit and with approves from owners

PRP:

- There is fixed amount of PRP in the system - 50,000,000 units. It is guaranteed by condition in the token smart-contract source code
- PRP token smart-contract can be created only when all 50,000,000 are stored to specified addresses
- When creating PRP some of tokens (like 2,000,000 - 5,000,000) will be stored at special PRP Wallet. This wallet will be used to manually transfer PRP as bounty and for other needs. All remaining PRP will be stored at PRP token smart-contract and will be part of auction
- Pre-sale investors receive PRP from pre-sale smart-contract using function buyTokens()
- Investors can change PRP to PPR with guaranteed +50% value using function exchange() of sale phase 1 smart-contract
- All unused PRP are burned on pre-sale auction finishing

PPR:

- There is fixed amount of PPR in the system - 1,000,000,000 units. It is guaranteed by condition in the token smart-contract source code
- PPR token smart-contract can be created only when all 1,000,000,000 are stored to specified addresses
- It is required to have enough PPR at sale phase 1 smart-contract. It is guaranteed by condition in the pre-sale smart-contract source code. This amount of needed PPR is calculated based on chosen bonuses for private investors and pre-sale investors and guaranteed enough to perform all transactions
- When creating PPR some of tokens will be reserved for pre-sale & sale phase 1 auctions, some of tokens will be reserved for sale phase 2 auction, some for internal needs. It is not final decision about wallets used
- Investors can get PPR from sale phase 1 smart-contract using methods claim() and exchange(), but we need a way to perform that automatically for all participants
- PPR are transferred so when investors get PPR they can be transferred everywhere
- At any moment owners of one of PPR Wallets can manually transfer PPR to any address
- All unsold PPR during sale phase 1 auction are stored to one of PPR Wallet and will be selling at another auction in 30 days