Dune-FreeTON Swap Architecture

1. Abstract

This is a submission for proposal #149 Contest: Dune → FreeTON Swap Architecture [1], the first phase of the implementation of proposal #117 Dune Network Merger [2] voted on Feb 1, 2021. We describe an infrastructure containing a website, where Dune users can easily swap their tokens to FreeTON, a set of independent relays/oracles monitoring both networks and propagating information between them, a Love contract on Dune Network to record swap requests and lock their tokens, and a set of Solidity contracts on FreeTON to commit the swap and transfer the tokens.

Table of Contents

2. Introduction

This is a submission for proposal #149 Contest: Dune → FreeTON Swap Architecture [1], the first phase of the implementation of proposal #117 Dune Network Merger [2] voted on Feb 1, 2021.
From the proposal, during the merge between Dune and FreeTON, Dune users will be able to swap their DUN tokens (the current supply is 458 M DUN) at a rate of 50.3 DUN for 1 TON, giving an estimated supply of 9.2 M TON. No vesting will be done, except for the TON tokens swapped with the 24M of DUN of Origin Labs, that will be vested for 16 months.
We describe an architecture for the swapper that is adapted to the sharding principle of FreeTON (not centralized in one smart contract) and uses a set of relays/oracles to protect against a subset of malicious relays. The FreeTON half of the architecture is generic enough, so that it can easily be reused for other mergers in the future.

Wallet

Contact on Telegram: @fabrice_dune
Surf wallet: 0:60a74bf0a86b3ab44de42b8ccac944ff8fa95745add04b2505cb84fd42265783
Public Key: 0x97d1ac029229af5c5e7196932106186a4a085fe988562409803b3c4508da5475

3. Requirements

The following requirements should be satisfied by this architecture:

- 50.3 DUN tokens are exchanged for 1 TON token
- The Origin Labs 24 M DUN tokens should be locked as a condition before the swap is started, and exchanged against TON tokens vested in a DePool contract for 16 months
- When a Dune user submits a swap by locking his DUN tokens on Dune, the corresponding TON tokens should automatically be locked on FreeTON with the same secret:
    - The user has a delay to provide his secret on FreeTON (we propose one week)
    - If he does not provide the secret, the TON tokens are automatically unlocked and returned after the delay; After another delay, the user can unlock and retrieve his DUN tokens (we propose a second week). He may retry to swap them again later, before the end of the merger period.
    - If he does provide the secret on FreeTON during the delay, the TON tokens are transferred to the address he has provided for the swap. The secret should be automatically used on Dune to lock his DUN tokens forever.
- No single entity should be trusted to propagate information between the 2 networks. Instead, the infrastructure should work with a set of independent relays, among which a minimal number of confirmations are required to confirm every swap (2 on 3, or 3 on 5, etc.). A specific contest might be used to select such relays.

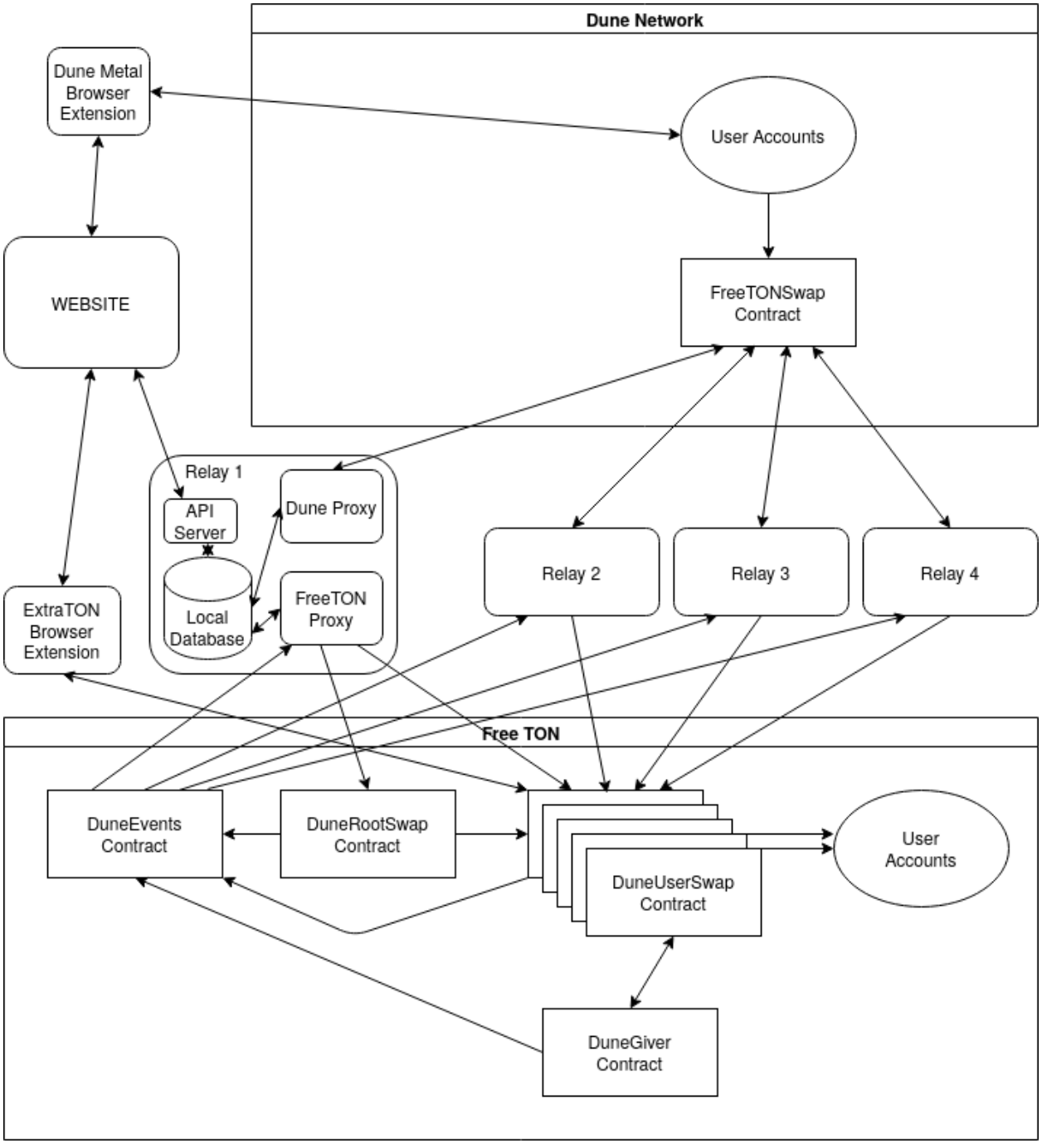
- The infrastructure should support adding or removing relays, and changing the minimal number of confirmations required.
- At the end of an expiration date, it should be impossible to submit new swaps on Dune.
- At the end of the expiration date plus some delay (we propose 3 weeks), all TON tokens that have not been transferred to user accounts or DePool should be returned to the main giver of FreeTON. All contracts should be killed (selfdestruct).
- The infrastructure should run at minimal cost. In particular, a distributed (TIP-3 like) architecture should be used on FreeTON instead of a centralized contract.
- The gas spent by the infrastructure should either be taken from swap orders or swap orders should have a minimal amount, to prevent attempts to spam the FreeTON network (we propose to have such a minimum around 1 TON per swap order)

4. General Architecture

We propose the following architecture to satisfy these requirements (also presented in the figure below):

- A website is used by Dune holders to perform the swap. The website is connected to Dune Metal browser extension (to transfer the user's tokens), to the ExtraTON browser extension (to reveal the secret on FreeTON), and to one or more relays/oracles to watch the status of the swap orders;
- A Love smart contract in Dune Network receives the swap orders and, if the secret is revealed, locks the tokens forever;
- A set of relays/oracles propagate information between Dune Network and FreeTON. They use:
    - A local database to keep their state persistent (though they can be restarted at any time from an empty database);
    - An API server to provide information to websites watching the merger;
    - A Dune proxy that crawls the Dune blockchain and registers new swap orders in the database. If a secret is present in the database, it is used to lock funds in Dune.
    - A FreeTON proxy that crawls the FreeTON blockchain and stores revealed secrets in the database. If a new swap order is present in the database, it is propagated on FreeTON, with the requirement that a minimal subset of the relays have confirmed it
- A set of Solidity smart contracts in FreeTON:
    - A DuneGiver smart contract keeps the supply of TON tokens and only send them when swap orders have been confirmed;
    - A DuneRootSwap smart contract manages the list of relays and is used to deploy and initialize DuneUserSwap contracts;
    - A DuneUserSwap contract is used to perform all swaps associated with a given public key. If a swap is confirmed by enough relays, the corresponding amount of TON tokens is retrieved from the DuneGiver and send to the user account on FreeTON;

- A DuneEvents contract is used to monitor everything. It receives messages from other contracts and emit events that are watched by the FreeTON proxies of the relays



5. Website - UX

In order to allow the users to easily swap tokens, they should be able to use a dApp. Here is a description of the workflow.

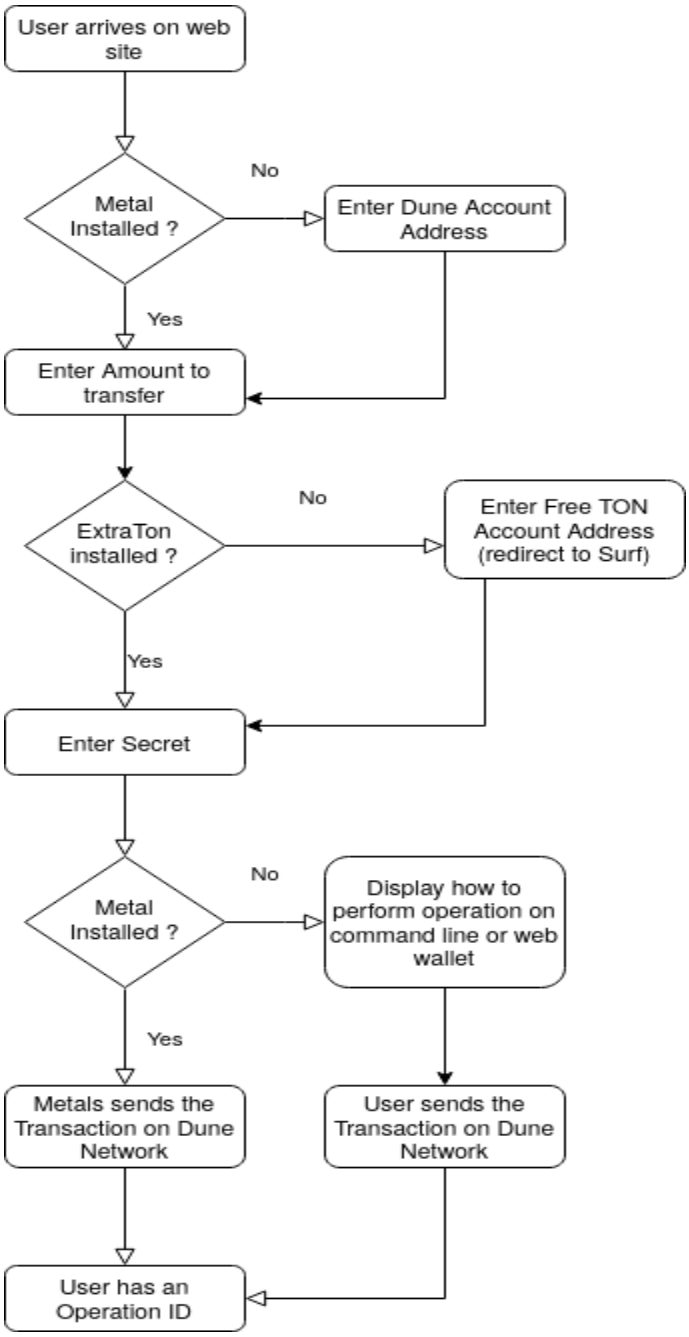
5.1. Commitment (Dune Network side)

The first step is to deposit dun tokens in the smart contract.

For this step the dApp must permit:
- the choice of an amount of dun tokens to be committed
- the choice of a secret which will be later used to finalized the swap
- the choice of a valid address on Free Ton chain where the user will receive the Ton Crystal tokens
- the choice of a valid address on Dune Network chain where the amount will be refund in case of expired swap (secret not revealed)

This is the initialisation of the swap procedure, the dApp will call the entrypoint `deposit` of the FreeTONSwap contract on Dune chain.

The dApp should make use of the Dune Network browser wallet Metal (see



https://chrome.google.com/webstore/detail/dune-metal/halmgkimboipdmlbhmfpaagfkpjecklm) to call the contract on Dune or give the user the necessary directions to use the dune client.

5.2. Reveal (Free TON side)

The last step is the revelation of the secret on the Free TON chain.

For this step the dApp should display the list of commitments waiting to be revealed for the Dune account of the user (retrieve with Metal or a simple form with an address field).
When the user is ready to reveal the secret, the dApp will offer a summary of the actions that will be taken once the operation is confirmed.

This is the finalization of the swap procedure, the dApp will call the entrypoint `revealOrderSecret` of the DuneUserSwap contract on Free TON chain.
Since the secret can be revealed by anybody, the dApp can offer the possibility for the user to delegate this part to the relay. This means the dApp will call a relay API endpoint with the secret.

This dApp should make use of the Free Ton browser wallet Extraton (see https://chrome.google.com/webstore/detail/extraton/hhimbkmlnofjdajamcojlcmgialocllm) to call the contract on Free Ton or give the user the necessary directions to use the free ton client.

5.3. Miscellaneous

At any time the dApp should display the history of swap regardless of state (finalized / waiting for revelation).
The history should give the user an easy way to see the operations on each blockchains (e.g dunscan/ton.live links). In particular, for swaps in 'waiting revelation' state:
- dune explorer link to the deposit operation
- free ton explorer link to the DuneUserSwap contract

For finalized swaps:
- dune explorer link to the transfer to the burn contract
- free ton explorer link to the transfer of the DuneUserSwap funds

6. Relays Architecture

The relays, or oracles, are in charge of monitoring both blockchains, and propagate information between them : swap orders are propagated from Dune to FreeTON, and secrets are propagated from FreeTON to Dune.
A relay is divided in 3 components:

- The Local Database stores the current state of the proxy;

- The Dune Proxy monitors the Dune blockchain for changes and upgrades the local database accordingly. It also monitors the local database and reveals secrets or expires swap orders accordingly;
- The FreeTON Proxy monitors the local database and creates swap orders on FreeTON accordingly. It monitors the FreeTON blockchain for revealed secrets and upgrades the local database accordingly;

6.1. Local Database

The local database stores 3 kinds of tables:

- Dune Internal Tables: tables used to store persistently the state of the Dune Proxy
- FreeTON Internal Tables: tables used to store persistently the state of the FreeTON Proxy
- Shared Tables: tables used to communicate information between the Dune Proxy and the FreeTON Proxy

Internal tables will be described in the proxies' sections. Here, we give an overview of Shared Tables.

The most important one is the '*swaps*' table. It contains the following columns:

- `id integer primary key` : the unique identifier created for this order. This is a global unique identifier;
- `dune_origin varchar not null` : the address of the creator of the order
- `dune_amount zarith not null` : the amount of DUN to swap
- `ton_amount zarith not null` : the corresponding amount of TON to swap
- `status swap_status not null` : the current status of the swap
- `hashed_secret bytea not null` : the sha256 hash of the secret
- `swap_timestamp timestamp not null` : the date of creation
- `refund_address varchar not null` : the address to which DUNs should be returned in case of expiration
- `freeton_pubkey varchar not null` : the public key of the FreeTON user
- `freeton_address varchar not null` : the address to which TONs should be transferred
- `freeton_depool varchar` : the address of a depool, if TONs should be vested
- `confirmations` : number of confirmations by relays
- `secret bytea` : the secret after revelation

6.2. Dune Proxy

The Dune proxy maintains the following information in the database:

- *Blocks*: the blocks crawled by the crawler (with information like the level, if it is included in the main chain, etc.)

- Block_operations: raw operations on the Dune swap contract associated with each block
- Operations: decoded relevant semantic operations (Deposit with associated generated swap id, Swap, or Refund), or queued operations to inject (Swap, with secret revelation)
- Swapper_info: internal state of the swap contract for efficiency
    - `admin` : an address on the Dune Network which is allowed to set/change the delegate of the contract
    - `swap_start` : a timestamp indicating when the contract starts accepting deposits (*i.e.* when users can make commitments)
    - `swap_end` : a timestamp indicating when the contract stops accepting new deposits (the Expiration_date)
    - `swap_span` : a timespan (in seconds) which defines the allowed time for users to reveal their secret after making their deposit (users will likely be allowed a period of 2 weeks to reveal their secret in the implementation, TBD)
    - `total_exchangeable` : the total amount (in DUN) that can be exchanged for TON crystals. This is simply a safeguard (should be 477.85M DUN for 9.5M TON crystals).
    - `vested_threshold` : the minimum amount (in DUN) that must be committed to be vested on Free TON (in a depool) for the funds to be exchangeable (will be initialized with 24M DUN)
    - `locked` : true if the swap contract is now locked

A crawler takes care of monitoring the chain and more specifically the swap contract and reflects every operation in the database. Some actions do not need to wait for the required 10 block confirmations (like registering a swap id for a deposit). If there are reorganisations at some level, part of the chain can be dropped to move to another head in the main chain, while taking care of marking operations crawler on the other chain as invalid.

An injector bot takes care of injecting secret revelations (with the swap entry point, see 7.3) when necessary (*i.e.* after the revelation is injected of Free TON).

6.3. FreeTON Proxy

The FreeTON proxy maintains the following Internal Tables in the Local Database:

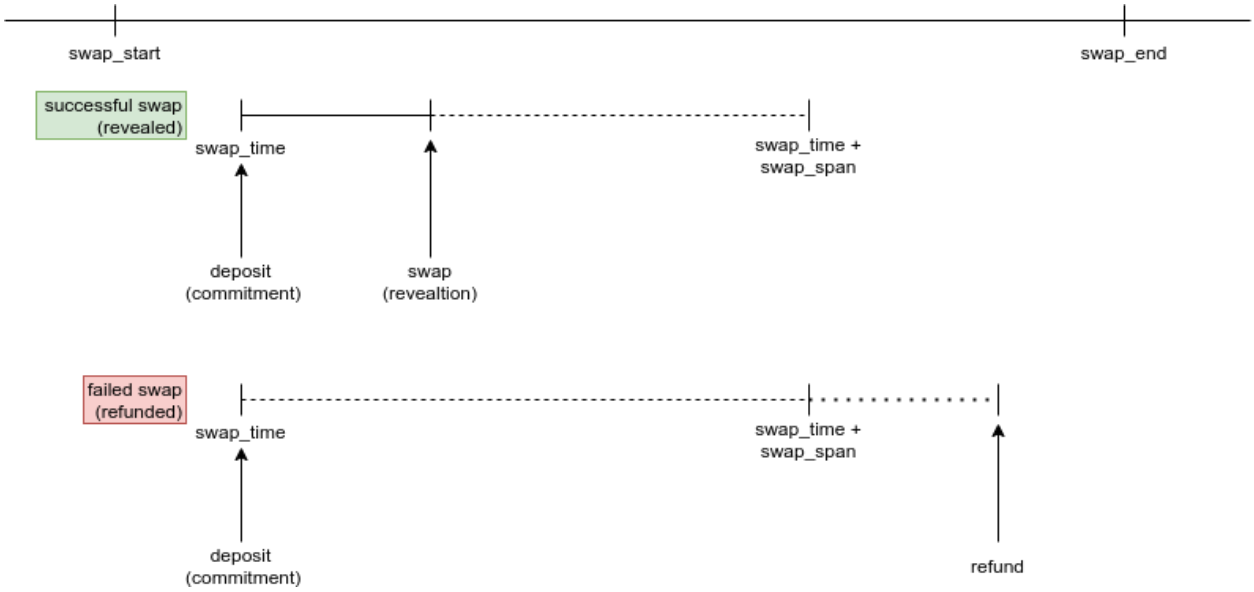
- freeton_contracts: a mapping between user public keys, creation dates and contracts addresses.
- freeton_state : some information that should be persistent for efficiency:
    - `giver_address`: the address of the DuneGiver contract
    - `root_address`: the address of the DuneRootSwap contract
    - `event_address`: the address of the DuneEvents contract
    - `last_blockid`: the last blockid for which a transaction query was performed and stored in the database (useful for fast restart of the relay after a stop)

7. Dune Network Architecture

Only one contract is deployed on Dune Network. It accepts commitments (*i.e.* deposits with a hidden secret) and secret revelations. Individual swaps are stored in a *big map* (a lazily deserialized hash table) which allows to scale independently of the number of ongoing swaps.

Because this contract is meant to receive and hold a very large amount of funds, it must be delegated to an active validator (baker). The proof of stake consensus mechanism of Dune Network allows one to delegate his/her staking (and voting) power to someone else without any transfer. If this contract is not delegated to an active validator, once it gathers a sufficiently large stake of the network, the blockchain could stop (or at least slow down considerably) which could prevent future swaps to be made in time.

Users are allowed to initiate swaps by an initial deposit and later reveal their secret to *finalize* their swap. They can be refunded *expired* non finalized swaps as shown in the timelines below.



7.1. Storage

The internal state of this contracts is initialized with the following *static* (immutable) information:

- `admin` : an address on the Dune Network which is allowed to set/change the delegate of the contract
- `swap_start` : a timestamp indicating when the contract starts accepting deposits (*i.e.* when users can make commitments)
- `swap_end` : a timestamp indicating when the contract stops accepting new deposits (the Expiration_date)

- `swap_span` : a timespan (in seconds) which defines the allowed time for users to reveal their secret after making their deposit (users will likely be allowed a period of 2 weeks to reveal their secret in the implementation, TBD)
- `total_exchangeable` : the total amount (in DUN) that can be exchanged for TON crystals. This is simply a safeguard (should be 477.85M DUN for 9.5M TON crystals).
- `vested_threshold` : the minimum amount (in DUN) that must be committed to be vested on Free TON (in a depool) for the funds to be exchangeable (will be initialized with 24M DUN)
- `locked` : true if the swap contract is now locked

Note: Deposits can be made between swap_stard and swap_end, while swaps can be finalized (revealed) between swap_start and swap_end + swap_span.

In addition to this static information, the contracts maintains the following:
- `swaps` : this is the hash table which stores the (ongoing and finished) swaps. This maps unique identifiers (increasing integer) to records of type `swap_t` (see 8.2. below)
- `swap_counter` : a simple integer counter incremented after each deposit to generate unique identifiers for swaps
- `total_vouched` : total amount in DUN currently committed for swap
- `vested_vouched` : total amount in DUN currently committed to be swapped for vested TON crystals on a depool
- `swapped` : total amount in DUN that has already been swapped (*i.e.* with revealed secret)

7.2. Swap structure

A registered swap is a record structure with the following information:
- `dune_origin` : the address who made the deposit on Dune Network
- `amount` : the amount of the swap in DUN
- `hashed_secret` : the SHA256 hash of the secret
- `time` : the timestamp at which the swap was initiated (*i.e.* the funds were deposited)
- `return_address` : the address (a public key hash) to return the funds if the swap in refunded
- `freeton_pubkey` : the public key of the FreeTON user
- `freeton_address` : the address to which TONs should be transferred
- `freeton_depool` : the address of a depool, if TONs should be vested
- `status` : the current status of the swap, either:
    - `Pending`, if the secret was not revealed yet, or
    - `Swapped`, if the secret was revealed (a swapped swap is final)

7.3. Entry points

The swap contract on the Dune Network has only four entry points:

- `deposit` : This entry point can be called by anyone who wishes to swap his/her DUN tokens (must be non zero). The user must provide the hashed secret, a return address in case of refund (which is not a smart contract), the address of the contract on Free TON that must be credited in accordance, his/her public key on Free TON. If the TON crystals are to be vested on Free TON, an address for the depool must be provided in addition. This entry point registers the new deposit with a new unique identifier (swap id).
- `swap` : This entry point is used exclusively to reveal the secret (and thus finalized the swap). The swap id, and the secret must be provided. It can be called by *anyone* who is in possession of the secret.
- `refund` : This entry point can be called by anyone to refund an *expired* swap. A swap is said to be expired if it is not finalized (*i.e.* the secret is not revealed yet) and the swap_span period is over for this particular swap (*i.e.* current time > swap_time + swap_span). Only the swap id must be provided for this entry point. A transfer is made to the swap's refund address and its amount is deducted from the various totals.
- `set_delegate` : This entry point is callable only by the admin to set the current delegate of the contract.
- `set_locked` : This entry point is callable only by the admin to lock the funds (prevent refunds) once the global swap period is over.

8. Free TON Architecture

4 main contracts are deployed on Free TON: the "DuneGiver" contract holds the TON tokens that will be exchanged during the merge; the "DuneRootSwap" contracts manages the official list of relays and creates "DuneUserSwap" contracts associated with each new user public key. "DuneUserSwap" contracts are used to manage all the swaps associated with a given public key, wait for confirmations by the minimal set of relays, retrieve TON tokens from the DuneGiver contract and transfer them to the user when he provides the secret for the swap. Finally, the DuneEvents contract receives important notifications from other contracts and emits events, so that relays watching it are notified of these events.

8.1. DuneGiver contract

DuneGiver is the contract that holds the TON tokens to be distributed during the merge. When a swap order is confirmed by enough relays, it transfers the corresponding tokens to the user swap contract so that the user can reveal his secret.

- `constructor(address freeton_giver)`, callable by Admin. Initializes the contract with the address to which unswapped TON tokens should be returned;
- `closeSwap()`, callable by Admin only 3 weeks after the merger expiration, calls `selfdestruct` and returns the unswapped TON tokens to the initial giver;

- `init(address root_address, address event_address)`, callable by Admin, performs the second phase of the initialization of the contract and of other contracts. Calls `getSwapInfo(event_address)` of DuneRootSwap `root_address`.
- `setSwapInfo(TvmCell duneUserSwapCode,uint64 expiration_date)`, callable by root_address. Called by the `getSwapInfo()` method of DuneRootSwap to perform the third phase of the initialization of the contract. Calls `init(root_address, duneUserSwapCode)` of the DuneEvents contract to finalize its initialization;
- `creditOrder(string order_id, uint128 ton_amount, uint256 pubkey)`, callable by DuneUserSwap contract associated with pubkey, calls `receiveCredit(order_id)` of its caller with the requested tokens, or `creditDenied(order_id)` if the balance is too low. This method can only be called by DuneUserSwap logic when the corresponding swap order has been confirmed by enough relays;

8.2. DuneRootSwap contract

DuneRootSwap is the contract in charge of deploying the user swap contracts when one of the relays demands it. It configures the user swap contract with the current list of relays pubkeys, so that they can confirm any order received. It also allows relays to change the current list of relays.

- `constructor(uint256[] relays, uint8 nreqs, TvmCell duneUserSwapCode, address giver_address, uint64 expiration_date)`, callable by Admin. First phase initialization of the contract, containing the list of relays' public keys, the required number of confirmation for a swap order, the code of the DuneUserSwap contract to be deployed, the address of the DuneGiver contract and the expiration date of the merger (one week is added to allow secret revelation);
- `closeSwap()`, callable by any relay, 3 weeks after the expiration. It returns the funds to the DuneGiver contract;
- `getSwapInfo(address event_address)`, callable by giver_address, performs the second phase of the initialization of the contract, and calls `setSwapInfo(duneUserSwapCode,expiration_date)` of the DuneGiven contract;
- `deployUserSwap(uint256 pubkey)`, callable by any relay, deploys the DuneUserSwap contract associated with the pubkey and calls the constructor its constructors with the current list of relays;
- `proposeChange(uint8 nreqs, uint256[] add_relays, uint256[] del_relays) returns (uint64 changeId)`, callable by any relay, to propose a change of the number of confirmations, a list of relays to add (by their public key) and a list of relays to remove. Returns the identifier to be used by other relays to confirm or reject the change. A change should be confirmed in 24h, or it is considered expired;

- `acceptChange(uint64 change_id)`, callable by any relay, to confirm a change proposal. If the number of confirmation is `nreqs`, the changes are performed.
- `rejectChange(change_id)`, callable by any relay, to reject a change proposal. If the number of rejections is `nreqs`, the change proposal is deleted;
- `cleanChanges()`, callable by any relay, to delete changes that have expired;
- `updateRelays(uint256 user_pubkey)`, callable by any relay, calls `updateRelays()` of the corresponding DuneUserSwap contract associated with user_pubkey to update its list of relays and number of confirmations;

8.3. DuneUserSwap contract

The DuneUserSwap contract is in charge of managing the swap orders sent for a particular public key address. It waits for an attempts to confirm an unknown order, wait for more confirmations, and when enough relays have confirmed the order, it requests the corresponding amount of TON tokens from the DuneGiver contract, waits for the secret revelation and then transfers the tokens to the destination address in the order. Users can declare a DePool for vested accounts. Orders can be cancelled when the expiration period is done.

- Static variables: address `s_giver_address` (DuneGiver) and address `s_root_address` (DuneRootSwap)
- `constructor(uint64 expiration_date, uint8 nreqs, uint256[] relays, uint8[] indexes, address event_address)`, callable by `s_root_address`, initializes the list of relays and number of confirmations needed by order;
- `closeSwap`(), callable by any relay 3 weeks after the expiration, refunds the DuneGiver contract;
- `updateRelays(uint256[] relays, uint8[] indexes, uint8 nreq)`, callable by `s_root_address`, when the list of relays has changed and this contract needs to be updated;
- `confirmOrder(string order_id, uint256 hashed_secret, uint64 ton_amount, uint64 dun_amount, address dest, address depool)`, callable by any relay, to create or confirm a swap order. When the number of confirmations is enough, calls `creditAccount`(...) of `s_giver_address` to receive the tokens;
- `receiveCredit(string order_id)`, callable by `s_giver_address`, when the tokens are credited to the contract. The user will be able to reveal its secret afterwards;
- `creditDenied(string order_id)`, callable by `s_giver_address`, when the DuneGiver does not have enough tokens on its balance;
- `cancelOrder(string order_id)`, callable by any relay after the expiration of the swap if the secret has not been revealed, can transfer back tokens to `s_giver_address`;

- `revealOrderSecret(string order_id, string secret)`, callable by User or any relay, can transfer tokens to user account, or calls `addVestingStake()` on a DePool contract;
- `receiveAnswer(uint32 errcode, uint64 info)`, callable by DePool contract, mandatory function if funds are transferred to a DePool. In case of error, put the state of the contract back so that `transferOrder()` can be called to retry the transfer;
- `transferOrder(string order_id)`, callable by User or any relay, can call `addVestingStake(...)` on a DePool contract, in the case where the previous transferred failed (because the donor was not registered);

Internally, a swap order is an automaton with the following states:

- WAITING_FOR_CONFIRMATION=0 : the swap order has not yet been confirmed by enough relays (initial state);
- CONFIRMED=1 : the swap order has been confirmed. A request should be made to DuneGiver to receive the corresponding tokens;
- WAITING_FOR_CREDIT=2 : a request has been sent to the DuneGiver to receive the tokens for the swap;
- CREDITED=3 : the tokens have been received from DuneGiver. The user should provide its secret to transfer the funds;
- REVEALED=4 : the secret has been revealed. The tokens can be transferred to the user account;
- TRANSFERRED=5 : the tokens have been transferred to the user. Unless the funds are vested and an error happens during the transfer, it is a final state;
- CANCELLED=6 : the order has expired before the revelation and the tokens have been transferred back to the DuneGiver contract;

8.4. DuneEvents contract

This contract receives messages from all other contracts when interesting events happen. Every method emits an event of the same name and parameters if the caller has been correctly verified. It can be monitored by relays to easily follow these events.

- `constructor( address giver_address )`, callable by Admin;
- `init( address root_address, TvmCell duneUserSwapCode )`, callable by giver_address, for final initialization;
- `userSwapDeployed ( uint256 pubkey, address addr )`, callable by DuneUserSwap contract, when the contract associated with a public key has been deployed by DuneRootSwap;
- `orderStateChanged( uint256 pubkey, string order_id, uint8 state )`, callable by DuneUserSwap contract, when the state of an order changed (see the different states in the previous section);

- `orderSecretRevealed( uint256 pubkey, string order_id, string secret)`, callable by DuneUserSwap contract, when the user correctly reveals the secret associated with an order;
- `orderCredited ( string order_id, bool accepted )`, callable by giver_address, when the tokens for an order have then credited to the corresponding DuneUserSwap contract;

9. Chronology

Assumptions:

- “freeton_giver” is the address of the contract that should collect all non-swapped tokens at the end
- A set of relays, “nreqs confirmations are required (for example, 3 of 5 relays)
- “Expiration_date” is the time at which swaps deposits cannot be injected on Dune anymore
- “Depool_address” is the address of the DePool contract on which Origin Labs tokens will be vested

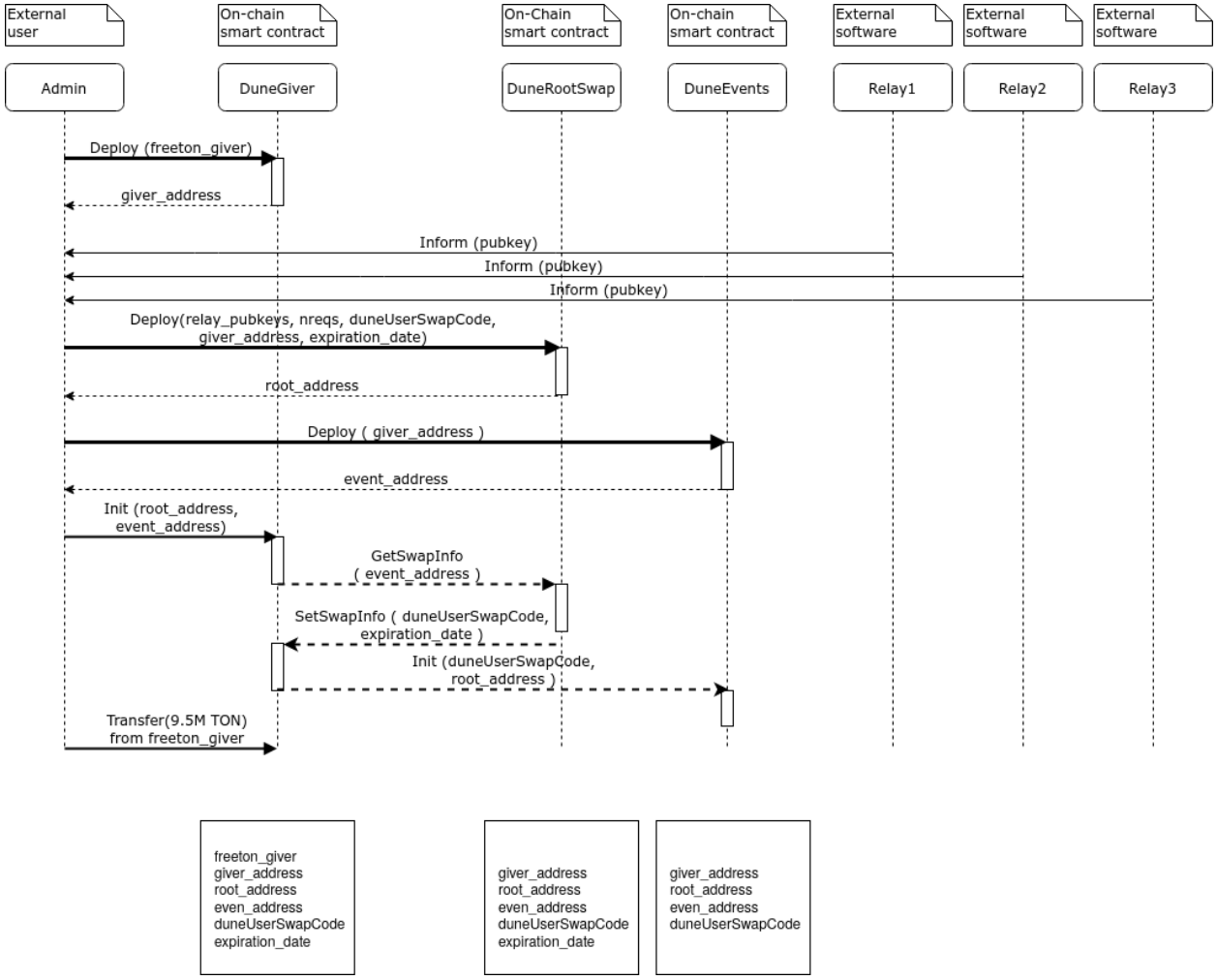
The following chronology should be used during the swap:

- Deploy the FreeTONSwap contract on Dune
- Collect public keys from all relays as “relay_pubkeys”
- Deploy the DuneGiver contract on FreeTON as “giver_address”, with argument freeton_giver;
- Deploy the DuneRootSwap contract on FreeTON as “root_address”, with arguments (relay_pubkeys, nreqs, duneUserSwapCode, giver_address, expiration_date,event_address)
- Call giver_address.init(root_address) to initialize the DuneGiver
- Origin Labs transfers 24M DUN to FreeTONSwap contract with the DePool address specified
- Enough gas should be transferred to DuneGiver and DuneRootSwap contracts for them to perform their work;
- Free TON transfers 9.2M TON to DuneGiver for the Dune community;
- All relays are started and Dune users can start swapping their tokens;
- Dune holders can swap their tokens on Dune Network until expiration_date;
- At expiration_date+3weeks, call the “closeSwap” method on all DuneUserSwap contracts and on DuneRootSwap contract to refund the DuneGiver;
- At expiration_date+3 weeks, call the “closeSwap” method on DuneGiver to refund the FreeTON giver of unswapped tokens;

10. Communication Diagrams

10.1. Sequence for FreeTON Initialization

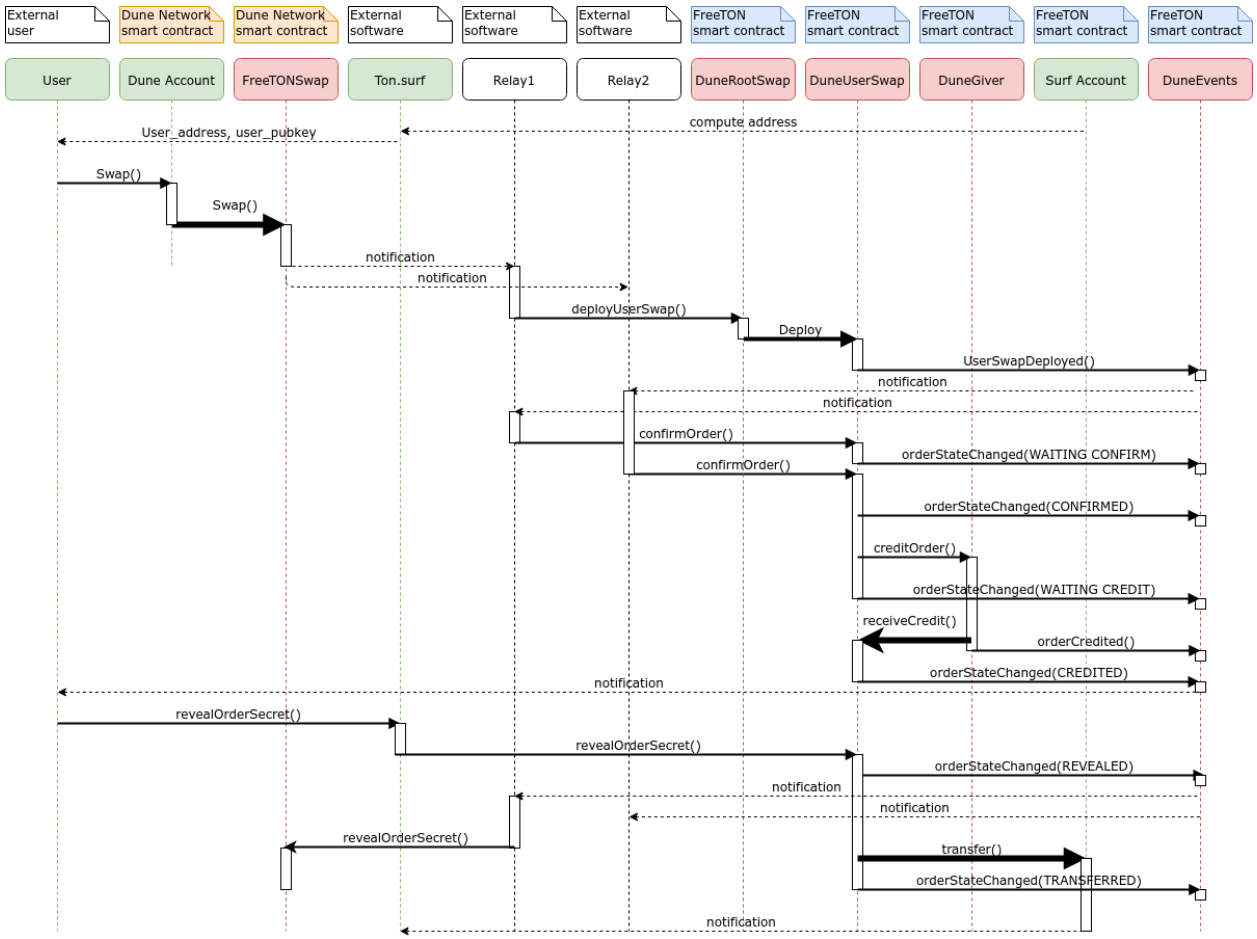
This is the initial sequence of operations on FreeTON to initialize the infrastructure of the swapper. At the end of this sequence, all contracts are initialized with the same data, especially the giver_address and root_address that are used to compute the address of every user swap contract and authentify all messages.



Initialization sequence on FreeTON

10.2. Sequence for a Successful Swap Order

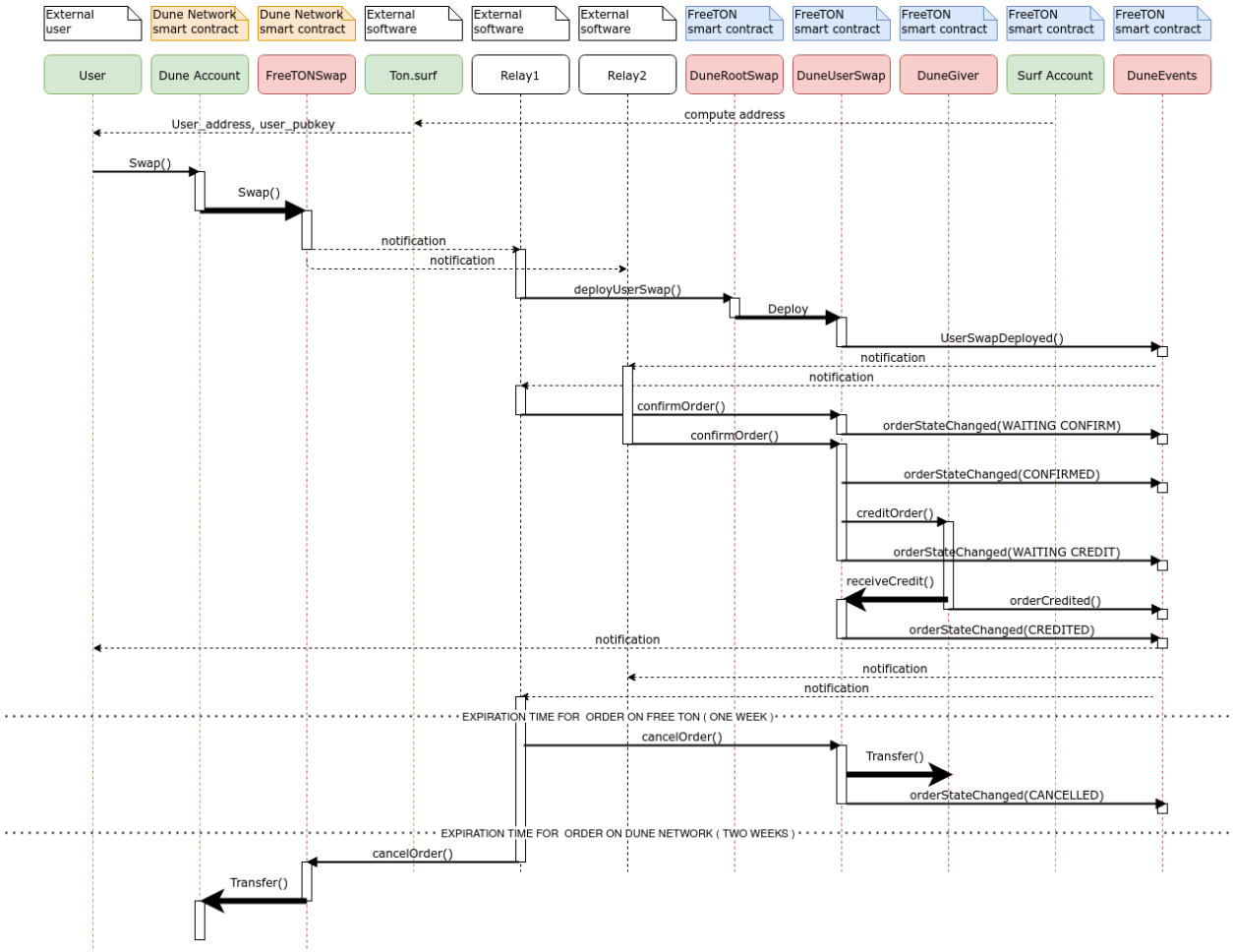
This sequence shows all the messages on the blockchains during a successful swap, from the user computing his FreeTON address to the tokens being transferred to that account.



Sequence for a successful swap (only 2 relays are visible, need 2 confirmations per swap order)

10.3. Sequence for a Failed Swap Order

This is the sequence of operations when a user fails to reveal his secret for the swap it initiated. At the end, the tokens on TON are sent back to the DuneGiver and the tokens on Dune are returned to the user refund address.



Sequence for failed swap because unrevealed secret

11. Development Tools

Website: OCaml[3] and OCaml to Javascript compiler [4]
Relays: OCaml [3], OCaml Postgres Binding pgocaml [5], ez_api API Server Library [6]
Dune Network: Liquidity smart contract language for Love [7], Spice testing framework [8]
FreeTON : Solidity compiler [9], OCaml FreeTON Wallet and SDK [10]

12. Links

[1] #149 Contest: Dune → FreeTON Swap Architecture:
https://gov.freeton.org/proposal?proposalAddress=0:cf1c6cbd204093c1a21ae5d384d47091020244da814f3c799dc346363ccc1afe
[2] #117 Dune Network Merger
https://gov.freeton.org/proposal?proposalAddress=0:0136a706eaf305dc824058dd942dad9982664bd2a7ca03b2143b247a220eff00

[3] the OCaml language https://ocaml.org/
[4] the js_of_ocaml OCaml to Javascript compiler
https://ocsigen.org/js_of_ocaml/latest/manual/overview
[5] OCaml Postgres Binding pgocaml https://github.com/darioteixeira/pgocaml
[6] ez_api API Server Library https://github.com/OCamlPro/ez_api
[7] Dune Network: Liquidity smart contract language for Love https://liquidity-lang.org
[8] Spice testing framework https://gitlab.com/o-labs/dune-spice
[9] Dune Metal browser extension https://gitlab.com/dune-network/dune-metal
[10] FreeTON : Solidity compiler https://github.com/tonlabs/TON-Solidity-Compiler/
[11] OCaml FreeTON Wallet and SDK https://github.com/OCamlPro/freeton-ocaml-sdk
[12] ExtraTON browser extension https://github.com/extraton/freeton