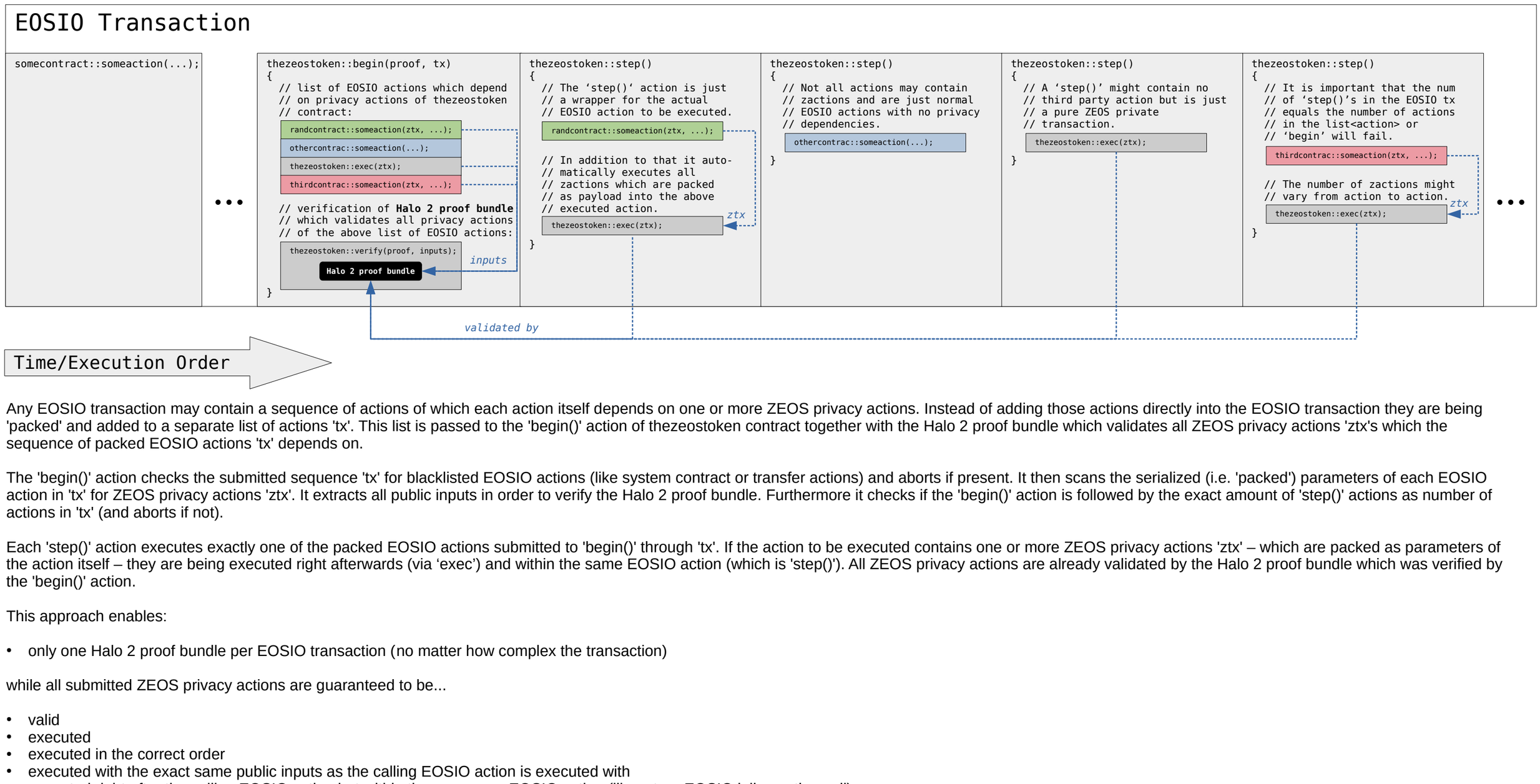
Concept

ZEOS takes full advantage of the Halo 2 proving system features like recursive proof bundling and verification (https://zcash.github.io/halo2/background/recursion.html). For complex EOSIO transactions which contain one or more actions which in turn depend on ZEOS privacy actions – so called ‘zactions’ – as inline actions (see graphics in whitepaper on p. 22 & 24) only a single Halo 2 proof bundle is required to validate all ZEOS privacy actions within the entire EOSIO transaction. For examples of complex protocol use cases requiring transactions to contain several ZEOS privacy actions see section 'Private Deposit & Withdrawal using the Proxy Account' of the ZEOS whitepaper (https://github.com/mschoenebeck/zeos-docs/releases/download/v1.0.0/zeos_whitepaper_v1.0.0.pdf).

EOSIO action parameters explained:
`tx:` list of EOSIO actions `(vector<action>)`
`ztx:` list of ZEOS privacy actions `(vector<zaction>)`
`proof:` Halo 2 proof bundle
`inputs:` public inputs to the proof bundle (each `zaction` in `ztx` comes with a set of public inputs which is required in order to verify the proof bundle)



Any EOSIO transaction may contain a sequence of actions of which each action itself depends on one or more ZEOS privacy actions. Instead of adding those actions directly into the EOSIO transaction they are being 'packed' and added to a separate list of actions 'tx'. This list is passed to the 'begin()' action of thezeostoken contract together with the Halo 2 proof bundle which validates all ZEOS privacy actions 'ztx's which the sequence of packed EOSIO actions 'tx' depends on.

The 'begin()' action checks the submitted sequence 'tx' for blacklisted EOSIO actions (like system contract or transfer actions) and aborts if present. It then scans the serialized (i.e. 'packed') parameters of each EOSIO action in 'tx' for ZEOS privacy actions 'ztx'. It extracts all public inputs in order to verify the Halo 2 proof bundle. Furthermore it checks if the 'begin()' action is followed by the exact amount of 'step()' actions as number of actions in 'tx' (and aborts if not).

Each 'step()' action executes exactly one of the packed EOSIO actions submitted to 'begin()' through 'tx'. If the action to be executed contains one or more ZEOS privacy actions 'ztx' – which are packed as parameters of the action itself – they are being executed right afterwards (via ‘exec’) and within the same EOSIO action (which is 'step()'). All ZEOS privacy actions are already validated by the Halo 2 proof bundle which was verified by the 'begin()' action.

This approach enables:

- only one Halo 2 proof bundle per EOSIO transaction (no matter how complex the transaction)

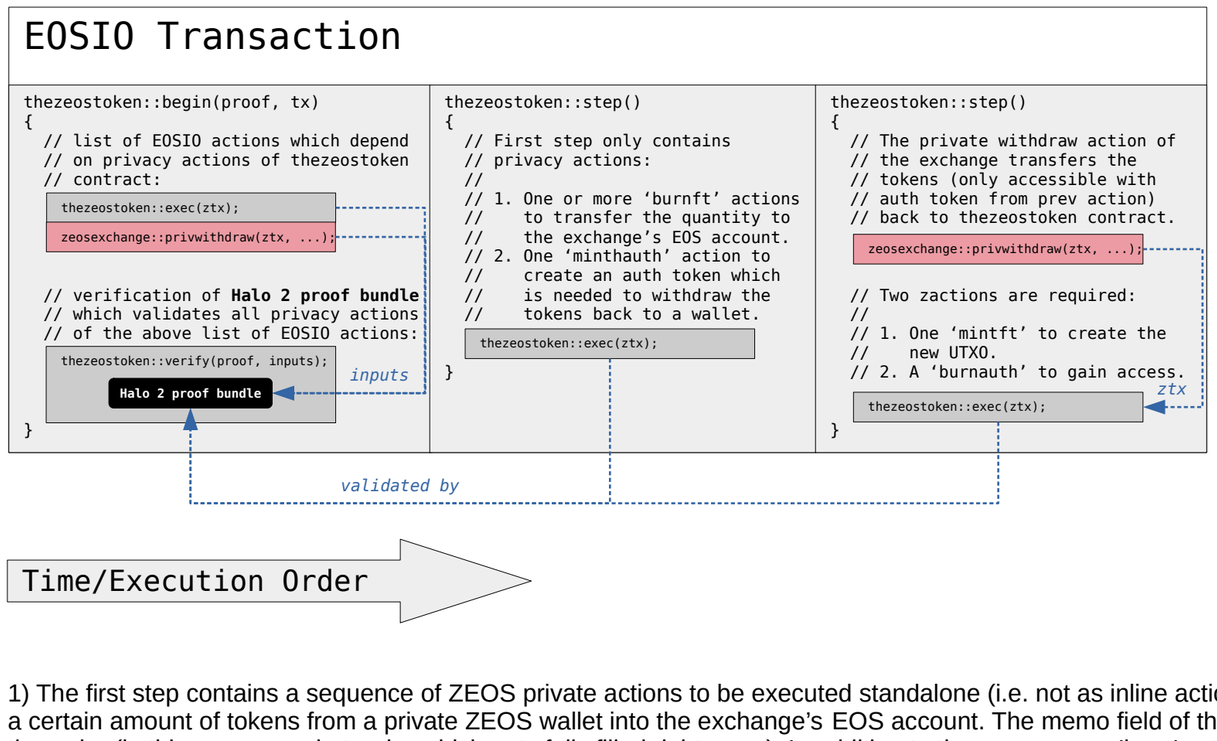
while all submitted ZEOS privacy actions are guaranteed to be...

- valid
- executed
- executed in the correct order
- executed with the exact same public inputs as the calling EOSIO action is executed with
- executed right after the calling EOSIO action but within the very same EOSIO action (like a true EOSIO inline action call)

A note on performance: Since all actions in 'tx' are already packed (i.e. having all their parameters serialized) they can be executed right away. The scanning for and extraction of the public input parameters for the proof verification involves zero memory copy but only minimal pointer arithmetic operations which makes this approach very efficient and highly scalable.

Example

Private market order execution on a DEX that implements ‘ZEOS Private Deposits & Withdrawals’ (see whitepaper page 22)



1) The first step contains a sequence of ZEOS private actions to be executed standalone (i.e. not as inline actions of a third party action). The privacy actions include burning one or more private UTXOs in order to transfer a certain amount of tokens from a private ZEOS wallet into the exchange’s EOS account. The memo field of the through 'burn' induced EOSIO token transfer contains the information required by the exchange to process the order (in this case a market order which gets fully filled right away). In addition to the one or more 'burn' actions the sequence contains a 'mintauth' action which mints an auth token required to privately withdraw the obtained tokens in the upcoming next step().

2) In the second step() the 'privwithdraw()’ action of the exchange contract is called in order to privately withdraw the obtained tokens into a private ZEOS wallet. To do so the exchange transfers the corresponding tokens (stored under the hash commitment's value of the previously minted auth token from the first step) back to thezeostoken contract. Afterwards the corresponding list of ZEOS privacy actions is executed which contains two zactions: One 'mint' which creates the new private UTXO representing the withdrawn tokens at the user specified private ZEOS wallet. Furthermore it contains one 'burnauth' that burns the auth token while at the same time authorizing access to the token that were just transferred in ‘privwithdraw()’.