ZEOS

#PrivacyMatters

TABLE OF CONTENTS

ABSTRACT

ZEOS enables private transactions for fungible and non-fungible assets on the EOS blockchain by leveraging Zcash privacy technology and LiquidApps' DAPP Network Services. No modifications of existing token contracts are required. Users are able to freely move their assets between transparent EOS accounts and private ZEOS wallets. Ownership of assets held in private ZEOS wallets is untraceable while hidden assets themselves are kept in custody of the ZEOS token contract. Ownership is represented by untraceable UTXOs which can be transferred privately among ZEOS wallets. UTXOs can be redeemed at any time in order to retrieve the underlying asset back into an EOS account. In addition to private transfers ZEOS will offer an easy-to-implement interface for all existing smart contracts on EOS to support private deposits and withdrawals. This enables all applications on EOS to become private-by-default protecting their user's privacy while staying fully transparent on the smart contract level for easy auditability. To completely unlink private transactions from users' personal EOS accounts the concept of a proxy account is introduced. This account has a special 'public' permission which can be used by anyone to conduct private transactions by paying a transaction fee denominated in ZEOS tokens. So-called 'Miners' compete for those fees by powering up the proxy account with EOS resources. Finally the ZEOS token powers a capital-efficient DeFi protocol for synthetic 'zAssets'. The entire ZEOS ecosystem is governed by smart organization[1].

1 As defined by Daniel Larimer

INTRODUCTION

The market for decentralized, permission-less finance – so-called 'DeFi' – is growing massively. Just in the past two years the total value locked in DeFi protocols went from around $600 million in 2020 to more than $230 billion in 2022. Crypto enthusiasts are well aware that DeFi will eventually replace most elements of the traditional financial system.

However, what almost all blockchains and decentralized applications in existence today completely lack is one of the most fundamental and basic human desires when it comes to financial matters: protection of user privacy. This is especially true for smart contract blockchains like EOS.

While it is a requirement for smart contracts of almost all applications to operate in full transparency – including all assets that go in and out – at least the identity of users interacting with them should remain anonymous. When it comes to peer-to-peer transactions between users, no sensitive transaction data – i.e. addresses of sender or receiver or asset type or amount – should be exposed publicly at all.

The current state of DeFi is an absolute nightmare from a privacy point of view. However, blockchain and DeFi have the potential to offer even stronger privacy for it's participants than the traditional financial system ever could. After all, in the current financial world no one is ever truly private: Even without government surveillance through regulation there is always a broker, a 'middleman' who facilitates an individuals access to financial markets. This ultimately means there is always somebody 'watching'.

In crypto, however, individuals are able to access markets directly, without an intermediary. This enables users – for the first time ever – to remain truly anonymous when acting on financial markets. This is just another potential benefit of DeFi over traditional finance which isn't even broadly acknowledged by the crypto community yet. True user privacy would be a game changer for DeFi and a major milestone in terms of maturity for the entire industry.

User privacy is not only desirable, but absolutely crucial for DeFi to succeed.

ZCASH PRIVACY

When it comes to privacy on blockchain, zero knowledge proofs have proven to be a powerful tool – particularly 'zk-SNARKs' – Zero-Knowledge Succinct Non-Interactive Argument of Knowledge. The development team of the cryptocurrency Zcash is spearheading this innovation by creating and maintaining arguably the most advanced and highly optimized open source codebase for zk-SNARK. Their code is widely used in a lot of cryptocurrency projects and reviewed by a lot of development teams. ZEOS utilizes the exact same codebase, specifically the zk-SNARK proving system to enable private transactions on EOS.

Zcash, just prior to the time this paper was written, had it's 5th major network upgrade called 'Orchard'. With it, a new SNARK proving system called 'Halo 2' was introduced. Among other improvements the new proving system does not require the so-called 'trusted setup' anymore.

The trusted setup is a ceremony in which trusted community members perform a multi-party computation to safely generate a random set of parameters for each SNARK's arithmetic circuit. It is those parameters from which proving and verifying keys are then derived. These are the key pairs required to later generate and verify zero knowledge proofs.

Although a trusted setup – if performed properly – can be considered very secure since only one of any number of participants needs to be honest, it has always been a source for FUD – fear, uncertainty and doubt. However, with the new proving system a different arithmetization method is used to create the proving/verifying key pairs which eliminates the need for trusted parameter generation and therefore all FUD related to the 'trusted setup' as well.

ZEOS adapts Zcash's new proving system 'Halo 2' and provides state-of-the-art privacy technology for all applications in the EOS ecosystem.

ZcashSapling

Note: The 'SNARK verifier' from Pomelo Season One was developed and deployed on the Kylin testnet for the ZEOS demo application (proof of concept) prior to the time this paper was written. It still uses the 'Groth16' proving system of Zcash 'Sapling' and will be updated to the new 'Halo 2' proving system before being deployed to the EOS mainnet.

Halo 2

DAPP NETWORK SERVICES

When it comes to dApp development on EOS the DAPP network services are a great addition to the ecosystem allowing for unlimited scaling and the lowest possible transaction costs for the end user. The following services are being utilized:

VRAM

In order to build a private UTXO transaction model based on zk-SNARKs inside an EOS smart contract, three ever-increasing data structures need to be maintained on chain. These data structures grow with each and every private transaction and are – except for adding new elements – read-only for the smart contract. Unfortunately the EOS blockchain RAM is expensive and rather unsuited for these large and mostly read-only data sets. VRAM on the other hand is perfectly suited for this task. It's a great solution for cheap and infinite storage which is directly accessible for smart contracts on EOS. Since all funds held privately using ZEOS are basically stored in VRAM it is noteworthy that the entire VRAM state can always be restored by replaying the EOS blockchain. Furthermore it is impossible to tamper with data in VRAM since this would be detectable on chain. These properties make VRAM truly trust-less.

VCPU

For the zero knowledge proof (ZKP) verification ZEOS utilizes the VCPU service as part of 'LiquidHarmony'. This might only be temporary since the proof verification should ideally happen on chain. However, in preparation for this paper, on chain proof verification based on the Groth16 proving system has been tested and evaluated on the Kylin testnet: An on chain verification of a simple proof of knowledge of a 32-byte hash seed takes around 150ms using a C++ implementation of Zcash's Saplings verifier linked to an EOS smart contract. Unfortunately this exceeds the 30ms execution limit of EOS transactions by multitudes – not to mention the resulting costs of such a long execution time. Full or at least partially on chain verification might become possible in the future though. Certain intrinsics could be added to the EOSVM to either improve the elliptic curve arithmetic of ZKP verification, or perhaps even an intrinsic function for entire ZKP verification. This could remove the VCPU dependency in future versions of the protocol. However, utilizing VCPU makes private transactions relatively cheap: The cost for an entire private transaction is only about 2ms of CPU (Kylin testnet). Without VCPU, ZEOS couldn't be built on EOS at this point in time.

Oracles

To retrieve real world price data from outside the EOS ecosystem the DAPP network's oracle service as part of 'LiquidHarmony' is utilized. Price oracles are required for minting and burning zAssets.

This paper is still a 'work-in-progess' rather than a final whitepaper. More iterations will follow. Community feedback is welcome and contributions are appreciated.

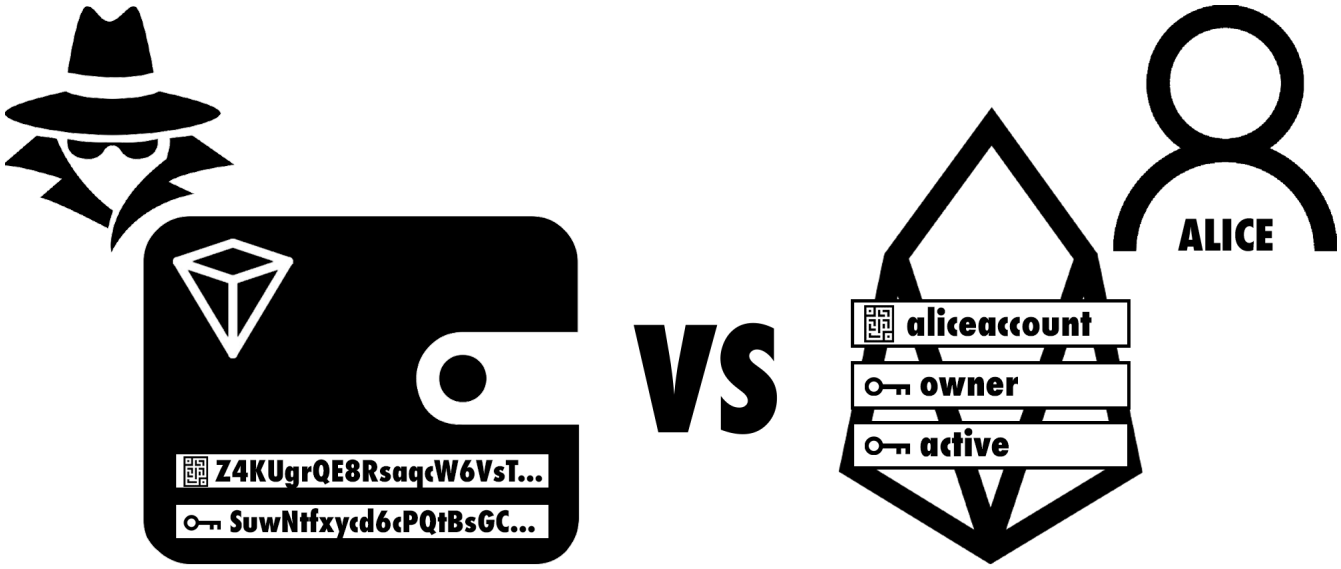
THE PROTOCOL

ZEOS is inspired by the Nightfall protocol for private transfers of fungible and non-fungible assets on the Ethereum blockchain. In it's original version though the protocol is very limited. ZEOS extends Nightfall's concept by adding useful features from Zcash. One example is Zcash's 'in-band secret distribution' which removes Nightfall's dependency on a secure side channel for private communication. This is required in order to share the secret transaction data with receivers of transactions. ZEOS eliminates the dependency of an additional communication channel and solely relies on the EOS public blockchain and the DAPP Network Services.

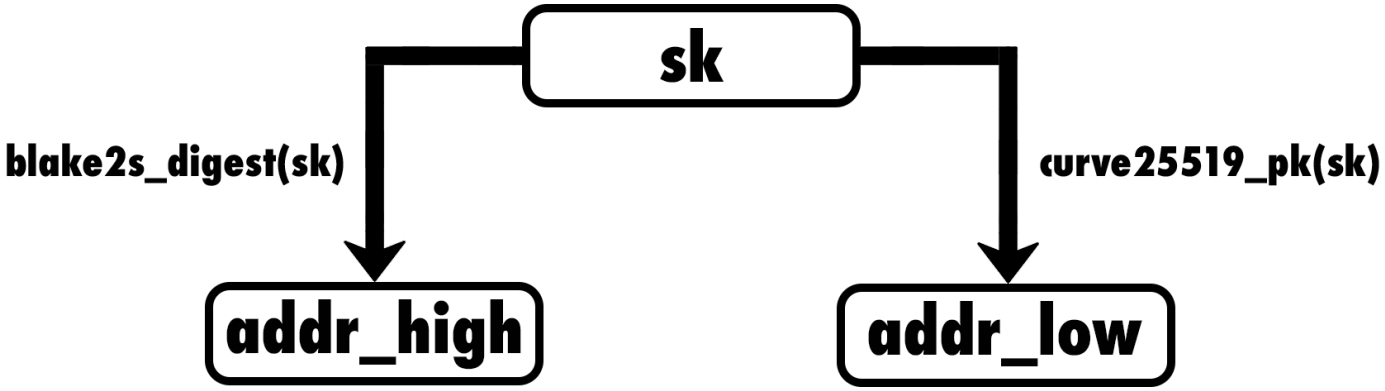
PRIVATE WALLETS

ZEOS introduces a new type of wallet where EOS assets can be held in full privacy. Private ZEOS wallets exist in parallel to transparent EOS accounts and assets can move freely between them.

Private ZEOS wallets are very similar to wallets of legacy cryptocurrencies like Bitcoin. Each wallet is defined through a secret spending key (aka private key). The receiving address of a wallet is then derived from that spending key. The same goes for ZEOS wallets. Creating a wallet basically means picking a 32-byte random number which represents the secret spending key – just like in Bitcoin.



In the current design the ZEOS address consists of two parts: The secret spending key's corresponding public key (based on curve25519) concatenated with the secret spending key's hash value (blake2s digest). This results in ZEOS addresses of 64 bytes length.

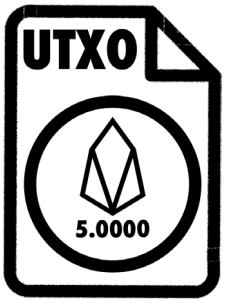


Each private ZEOS transaction comes with a unique 64-byte viewing key which reveals all details about the receiving part of the transaction. Viewing keys can be used by senders to prove a particular transaction has been executed and that certain assets have been received by a certain wallet.

PRIVATE UTXOS

Ownership of assets hidden in ZEOS wallets is represented through ownership of private UTXOs – Unspent Transaction Outputs. Besides fungible and non-fungible tokens ZEOS introduces a third token type which is non-transferable and represents a *private permission*. This kind of authentication token enables private deposits and withdrawals of fungible and non-fungible assets to and from any third-party smart contract in the EOS ecosystem.

Private UTXO types:

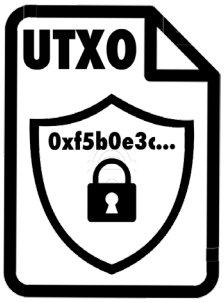


FUNGIBLE TOKEN

Represents a fungible asset on EOS. It is characterized by amount and symbol.

NON-FUNGIBLE TOKEN

Represents a non-fungible asset on EOS. It is characterized by a unique identifier.



AUTHENTICATION TOKEN

Represents a permission to access assets in custody of a certain smart contract. It is characterized by a hash commitment.

All UTXO types share the same data structure:

header: 64 bit
The header field contains meta information like the UTXO type and a number specifying under which protocol version it was created.

d0: 64 bit
The first data field contains either the amount of the particular asset (fungible token) or the lower 64 bits of an identifier (non-fungible token). In case of authentication token this field is undefined.

d1: 64 bit
The second data field contains either the symbol code of the particular asset (fungible token) or the upper 64 bits of an identifier (non-fungible token). In case of authentication token this field is undefined.

contract: 64 bit
In case of fungible and non-fungible token this field contains the EOS account name of the issuing token contract. In case of authentication token this field contains the EOS account name of the smart contract which issued the permission.

cool_down: 64 bit
Determines when this UTXO can be 'burned'. Specifically this value is an EOS block height from which this UTXO becomes 'burnable'. A cool down value enables several interesting features like burn delays for synthetic assets or private permissions which only become valid after a particular point in time (i.e. a particular blockchain height).

rho: 256 bit
A 32-byte random number (aka 'salt') to ensure unique hash values for UTXO commitments (fungible & non-fungible token). In case of authentication token this value is like a secret key which gives access to whatever the permission was created for.

The wallet address is not stored per UTXO but per transaction that creates the UTXO. All private transactions include a 256-byte memo field for private messages as well.

FLOW CHART LEGEND

Illustrates what happens on the EOS blockchain when a private transaction is executed. It shows exactly what sensitive information is exposed and what an observer would be able to detect.

Illustrates the equivalent from the perspective of the virtual ZEOS world. It shows exactly what sensitive information is exposed and what an observer would be able to detect.

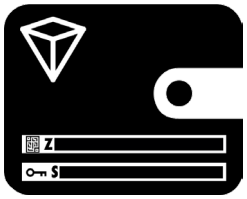
An EOS action which is initiated by a certain actor and/or executed by a certain EOS account and permission.

Private ZEOS transaction data that always contains a zero knowledge proof (ZKP) which proves the correctness of the private (hidden) inputs. All public inputs are exposed and observable by the public.

Signature of an EOS account permission

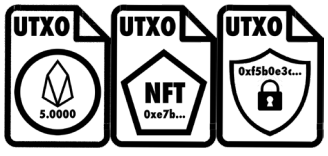
Signature of private transaction data which is basically the ZKP itself: If the proof is valid it automatically implicates the private transaction is signed by the correct secret spending key.

An EOS account with a certain account name (address) and certain permissions. If it is depicted with a '</>' at the top there is a smart contract deployed to this account. If there is no '</>' it is just a personal user account.



A ZEOS wallet with a certain address and secret spending key. No information about wallets is ever exposed by the protocol and is therefore always private.

Standard EOS fungible or non-fungible assets.



Unspent transaction output (UTXO) representing either a fungible or a non-fungible asset on EOS or a ZEOS authentication token.

Signing or calling an EOS Action.

Movement of assets/UTXOs between EOS accounts/ZEOS wallets.

Minting of UTXO

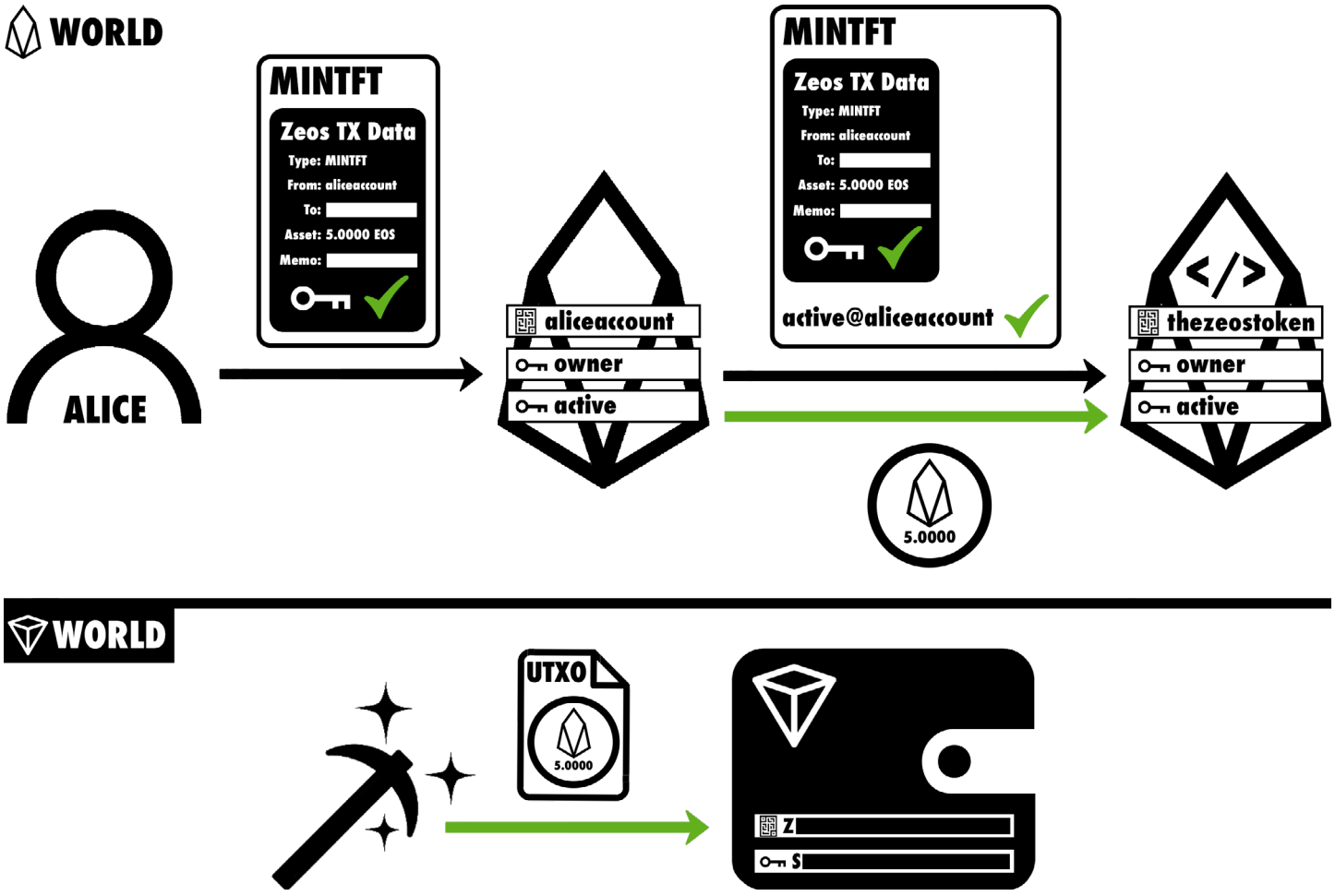
Burning of UTXO.

MINTING OF FUNGIBLE & NON-FUNGIBLE UTXOS

To move an asset from a transparent EOS account to a private ZEOS wallet the 'mint' action is called. To move an asset from a private ZEOS wallet back to a transparent EOS account the 'burn' action is called. For fully private and untraceable token transfers the 'ztransfer' action is called. These three actions exist for fungible as well as for non-fungible tokens:

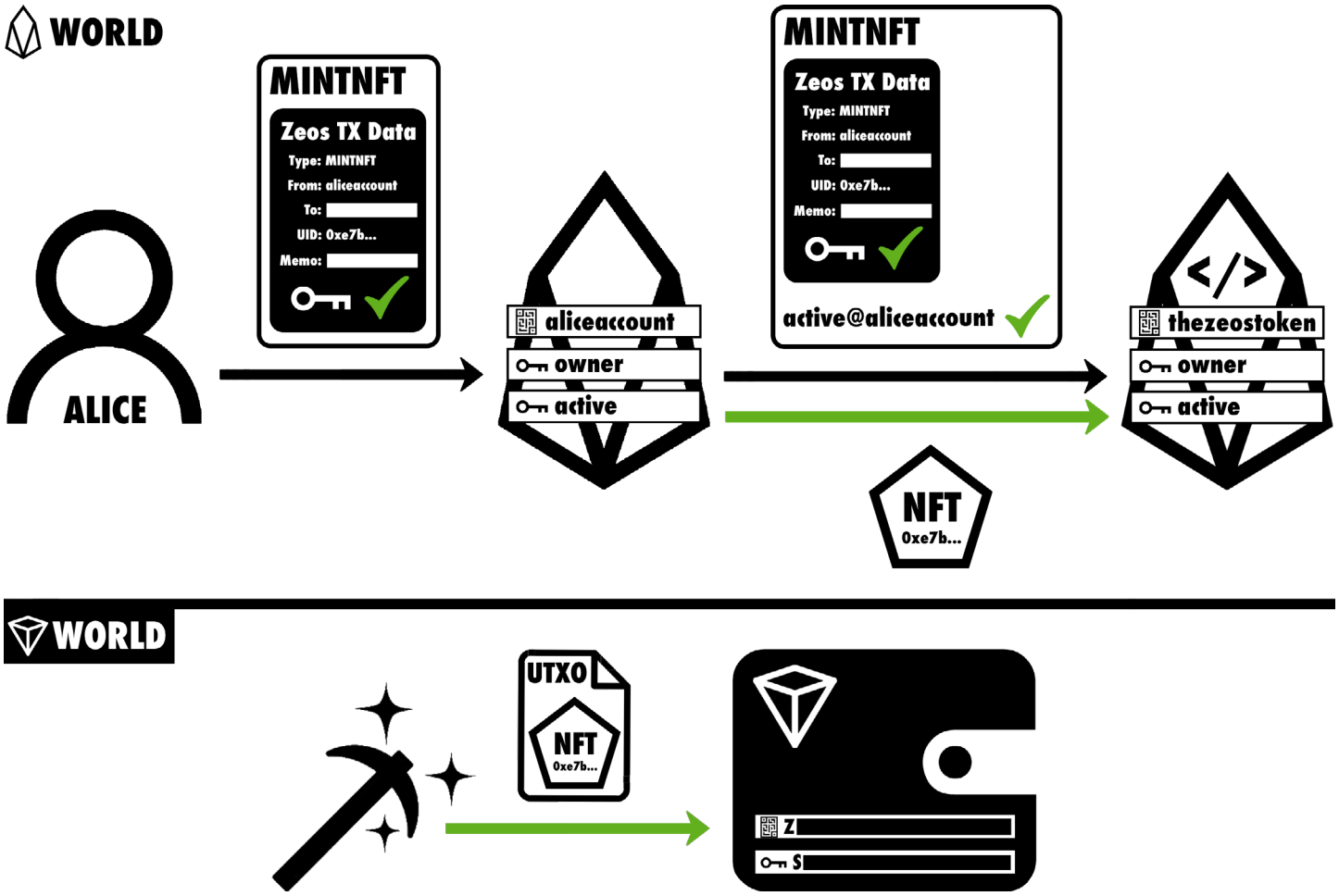
- 'mintft' and 'mintnft'
- 'burnft' and 'burnnft'
- 'ztransferft' and 'ztransfernft'

When 'mint' is called on a fungible or non-fungible token the specified asset moves from the user's EOS account into custody of the ZEOS token contract[2]. At the same time a private UTXO representing the asset is minted at the specified ZEOS wallet address. The information about which wallet address the UTXO is being minted at remains private. Only the asset's amount and symbol is publicly traceable when minting UTXOs. In fact, the protocol will never expose any ZEOS addresses publicly.



MINTFT: ALICE MOVES 5 EOS TOKENS FROM HER PUBLIC EOS ACCOUNT TO A PRIVATE ZEOS WALLET

2 Strictly, each call to 'mint' must be preceded by a 'transfer' call to move the specific

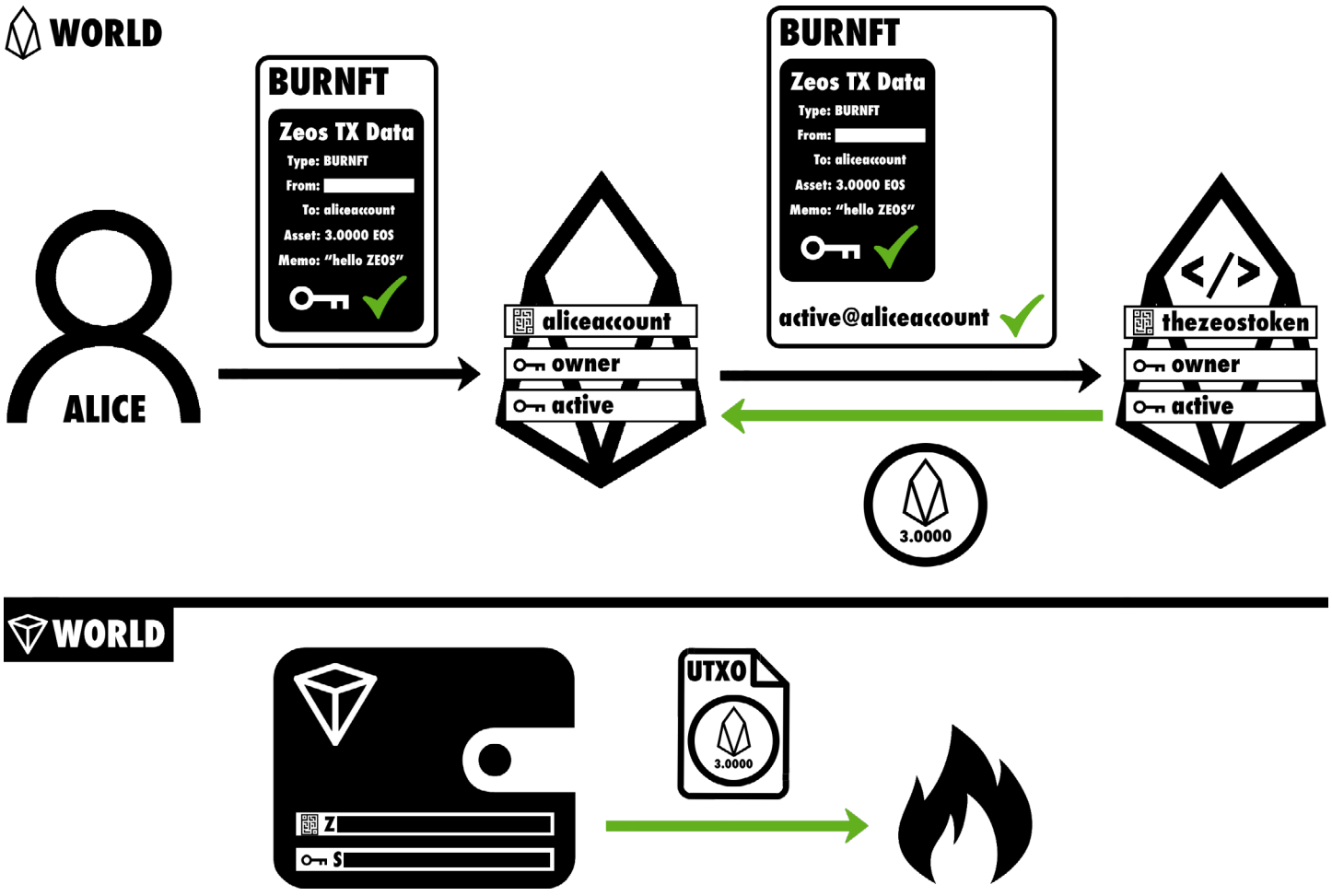


MINTNFT: ALICE MOVES AN NFT FROM HER PUBLIC EOS ACCOUNT TO A PRIVATE ZEOS WALLET

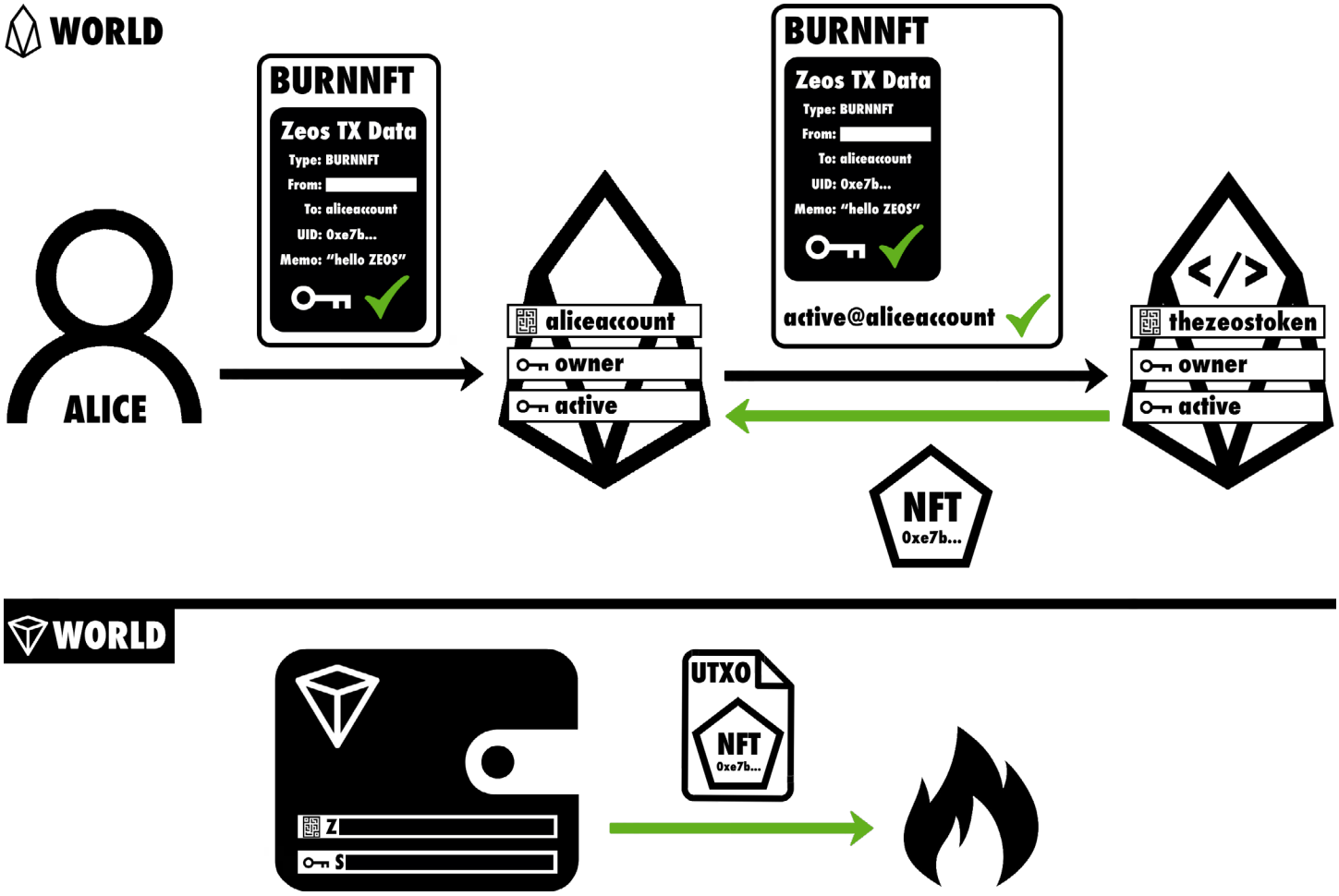
fungible or non-fungible asset from the user's EOS account to 'thezeostoken' account in compliance with the 'deposit/withdraw' concept of EOSIO smart contracts. For the sake of simplicity though this preceding transfer action is ignored in this paper.

BURNING OF FUNGIBLE & NON-FUNGIBLE UTXOS

At any time UTXOs are redeemable in exchange for the (same amount of) underlying asset they represent. To do so the UTXO is burned using the corresponding 'burn' action. When this happens the underlying asset is released from custody of the ZEOS token contract and is transferred to the EOS account specified by the user. The only limitation regarding burning could be a potential cool-down value any private UTXO might have. In this case the burn will fail until the particular EOS block height is reached. The only information which is revealed publicly when burning UTXOs is – again – only the asset's amount and symbol (or the unique identifier in case of NFT). The wallets address from where the asset is burned remains private.



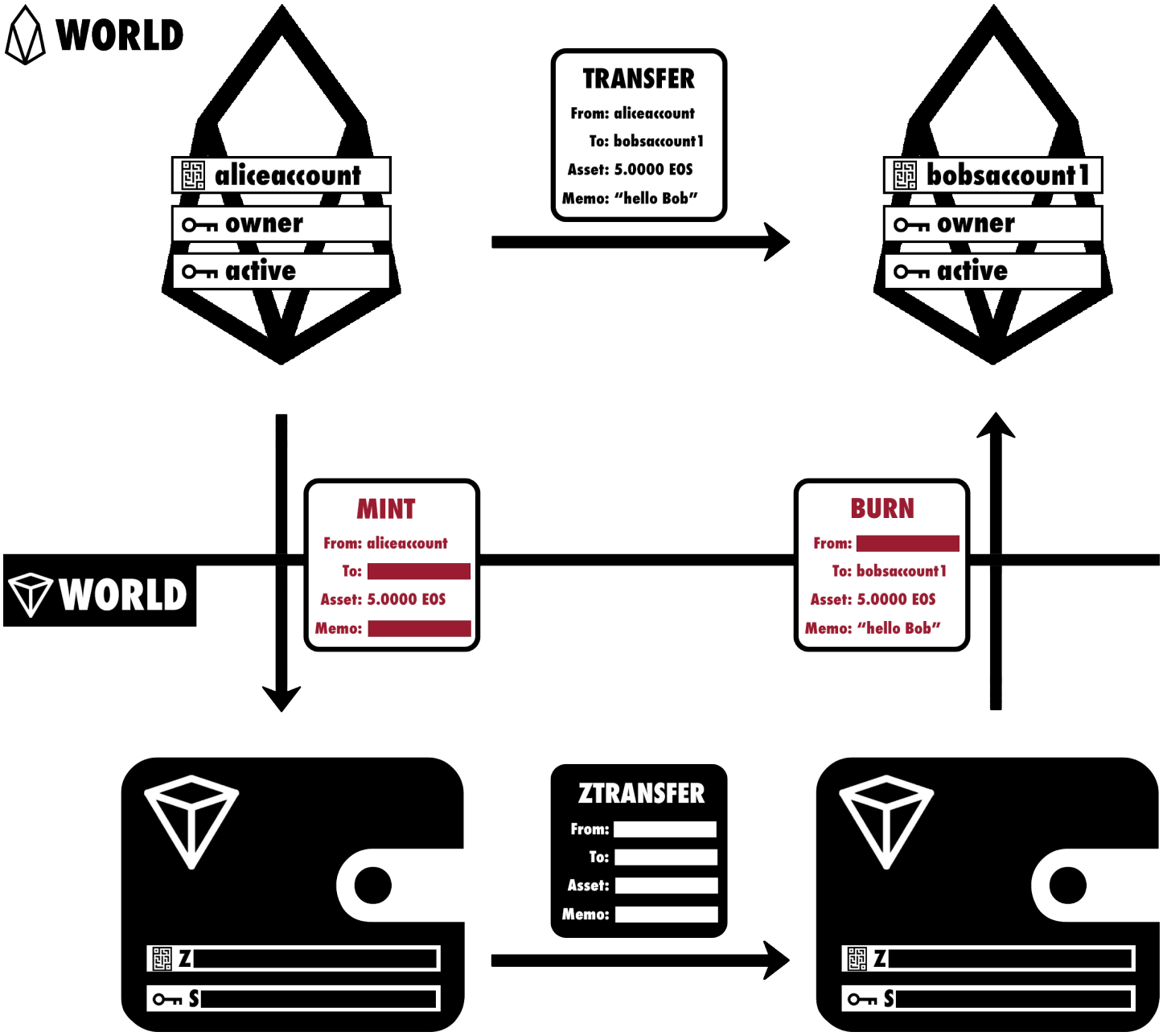
BURNFT: ALICE MOVES 3 EOS TOKENS FROM A PRIVATE ZEOS WALLET TO HER PUBLIC EOS ACCOUNT



BURNNFT: ALICE MOVES AN NFT FROM A PRIVATE ZEOS WALLET TO HER PUBLIC EOS ACCOUNT

CONNECTING THE TWO WORLDS

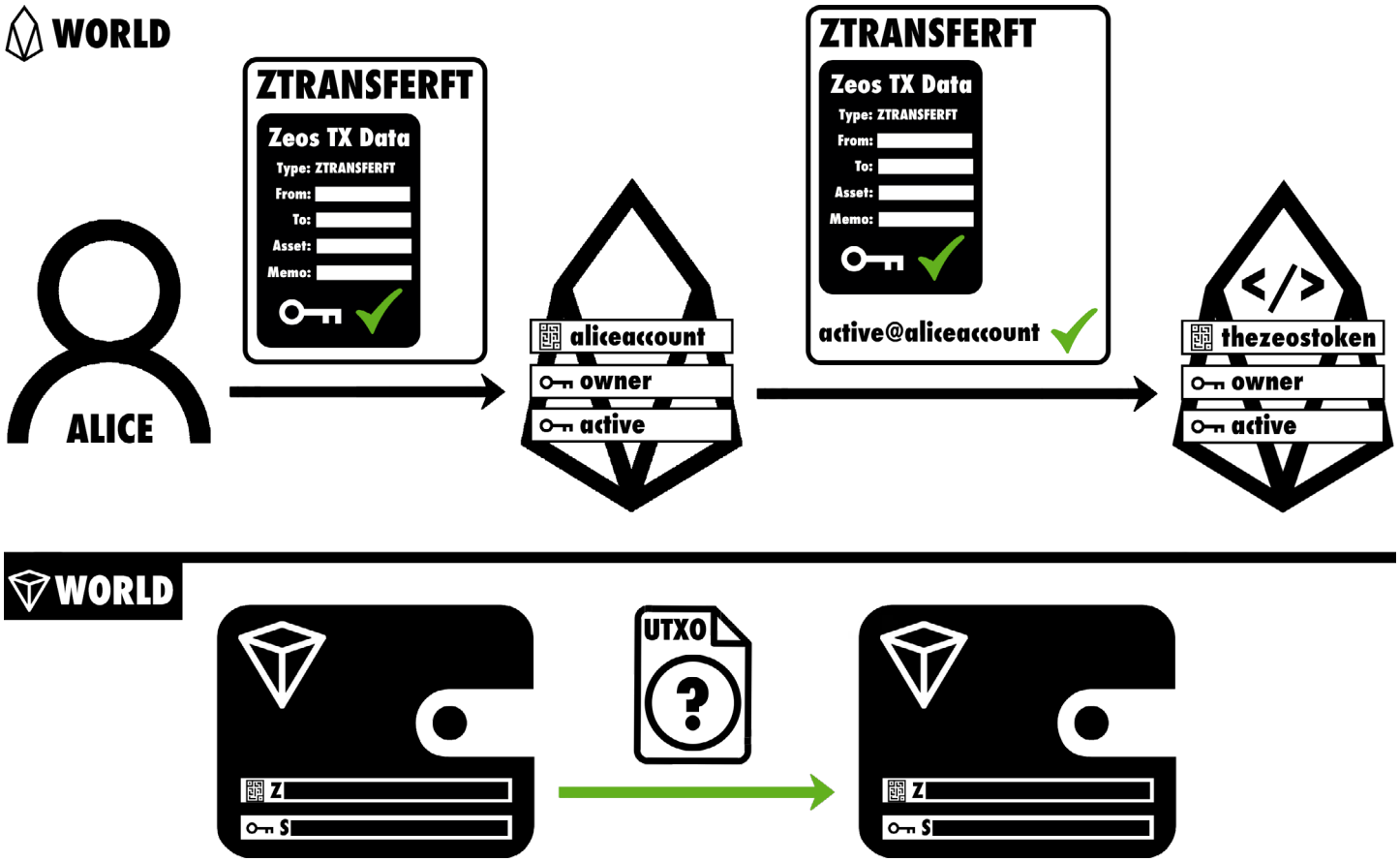
The 'mint' and 'burn' actions for fungible and non-fungible assets can be viewed as the gateway connecting the two worlds: The only way assets can move from the transparent EOS world to the private ZEOS world is by calling the mint action(s). And the only way assets can move from the private ZEOS world back to the transparent EOS world is by calling the burn action(s). For private peer-to-peer transfers the 'ztransfer' action for fungible or non-fungible tokens is being used in analogy to the EOS standard token 'transfer' action.



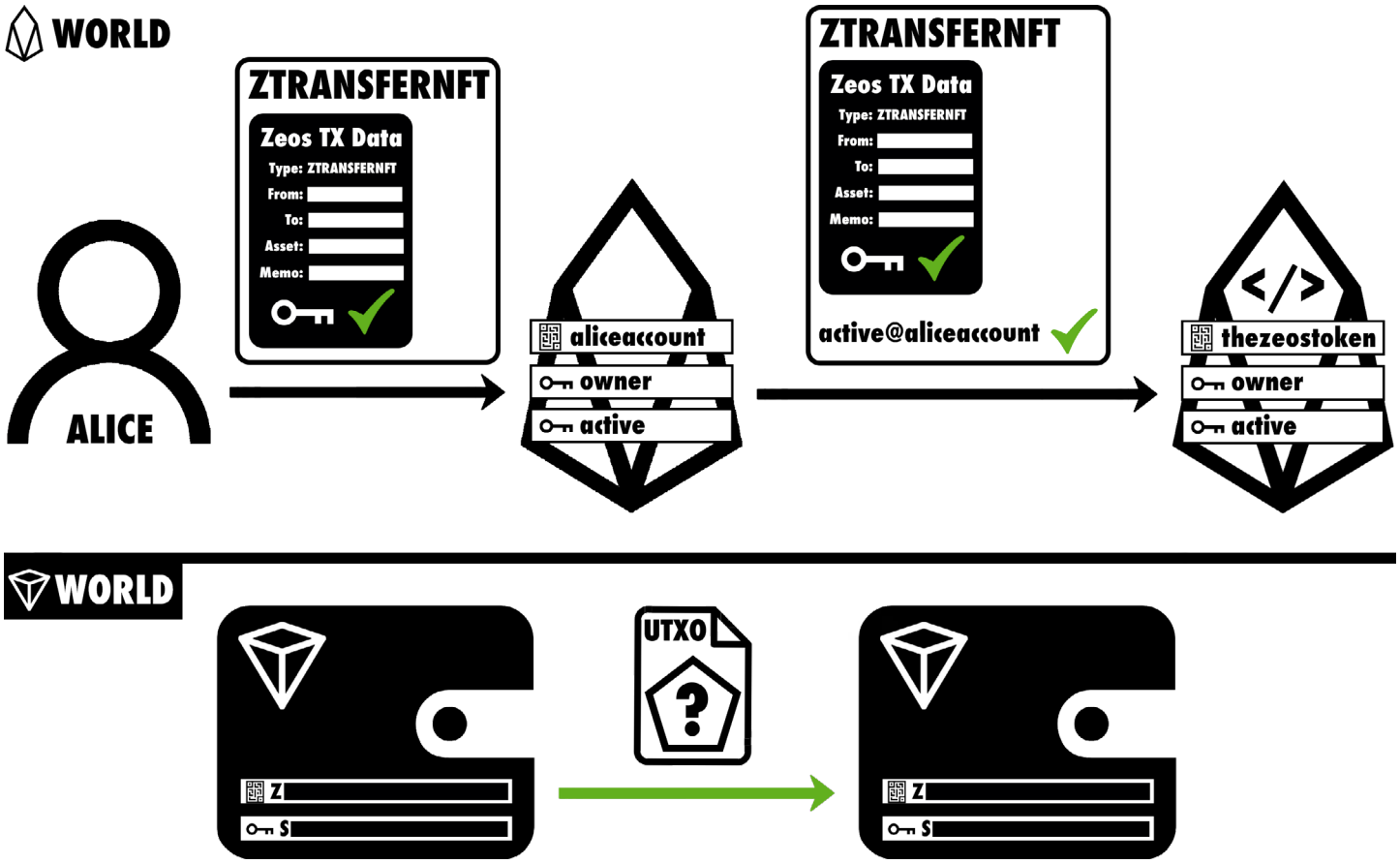
MOVING ASSETS BETWEEN THE 'TWO WORLDS' BY CALLING 'MINT' AND 'BURN' ON FUNGIBLE OR NON-FUNGIBLE TOKENS AND IT'S IMPLICATIONS FOR PRIVACY

TRANSFER OF FUNGIBLE & NON-FUNGIBLE UTXOS

UTXOs representing fungible or non-fungible tokens are fully privately transferable among ZEOS wallets. If an UTXO represents a fungible token it is divisible to the exact same degree as the underlying asset itself. UTXOs of non-fungible tokens are not divisible.



ZTRANSFERFT: ALICE CONDUCTS A PRIVATE TRANSFER OF FUNGIBLE TOKENS



ZTRANSFERNFT: ALICE CONDUCTS A PRIVATE TRANSFER OF A NON-FUNGIBLE TOKEN

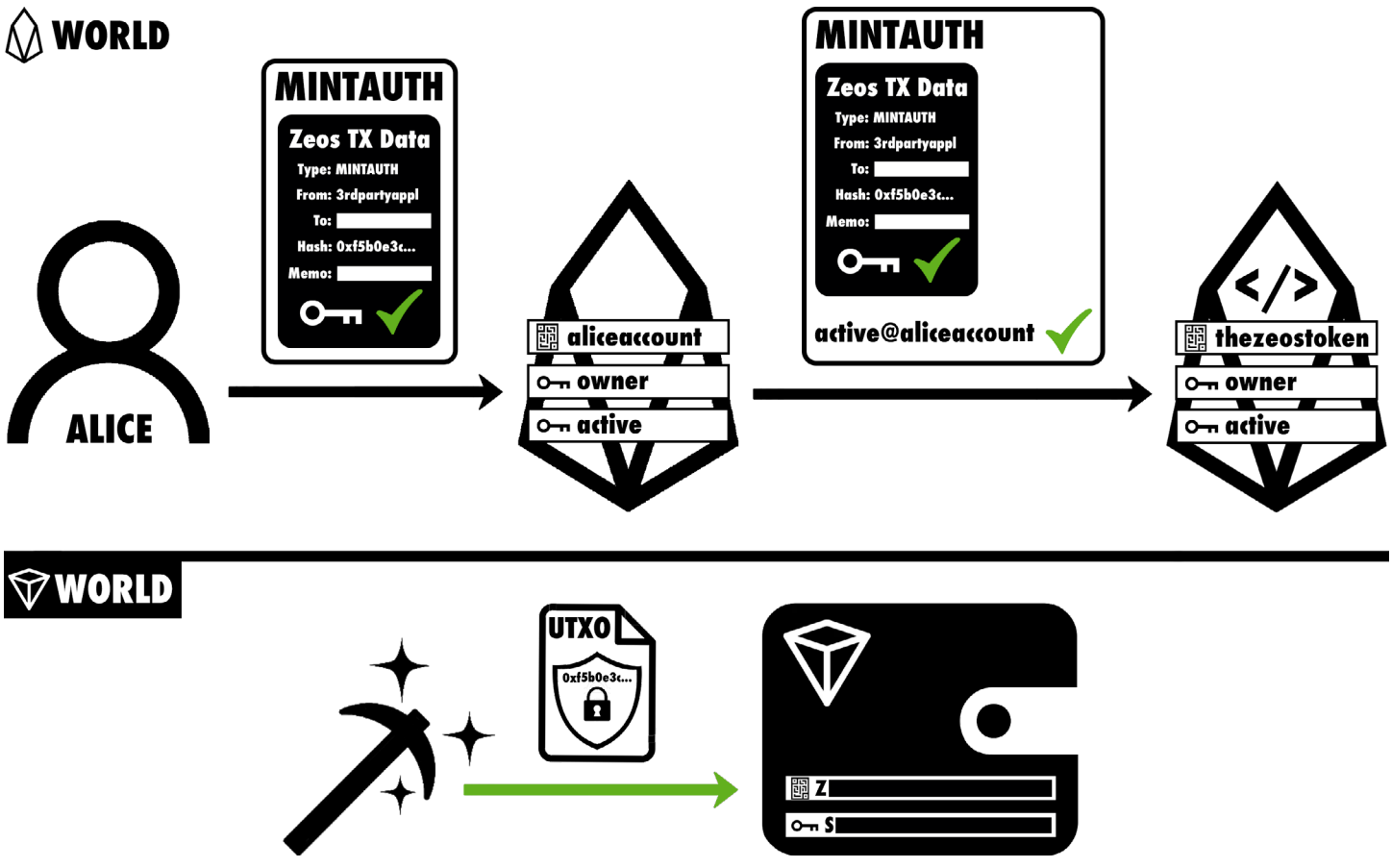
MINTING OF AUTHENTICATION UTXOS

Minting and Burning of authentication tokens does not move any assets directly. Instead an authentication token merely represents a permission which allows the holder to access assets of a particular smart contract. To mint an authentication token the user creates a hash commitment h such that:

$$h \stackrel{\text{def}}{=} hash(rho|addr)$$

where:
$hash()$ is a collision-resistant hash function
rho is a 32-byte random number
$addr$ is the wallet address where the token is minted at

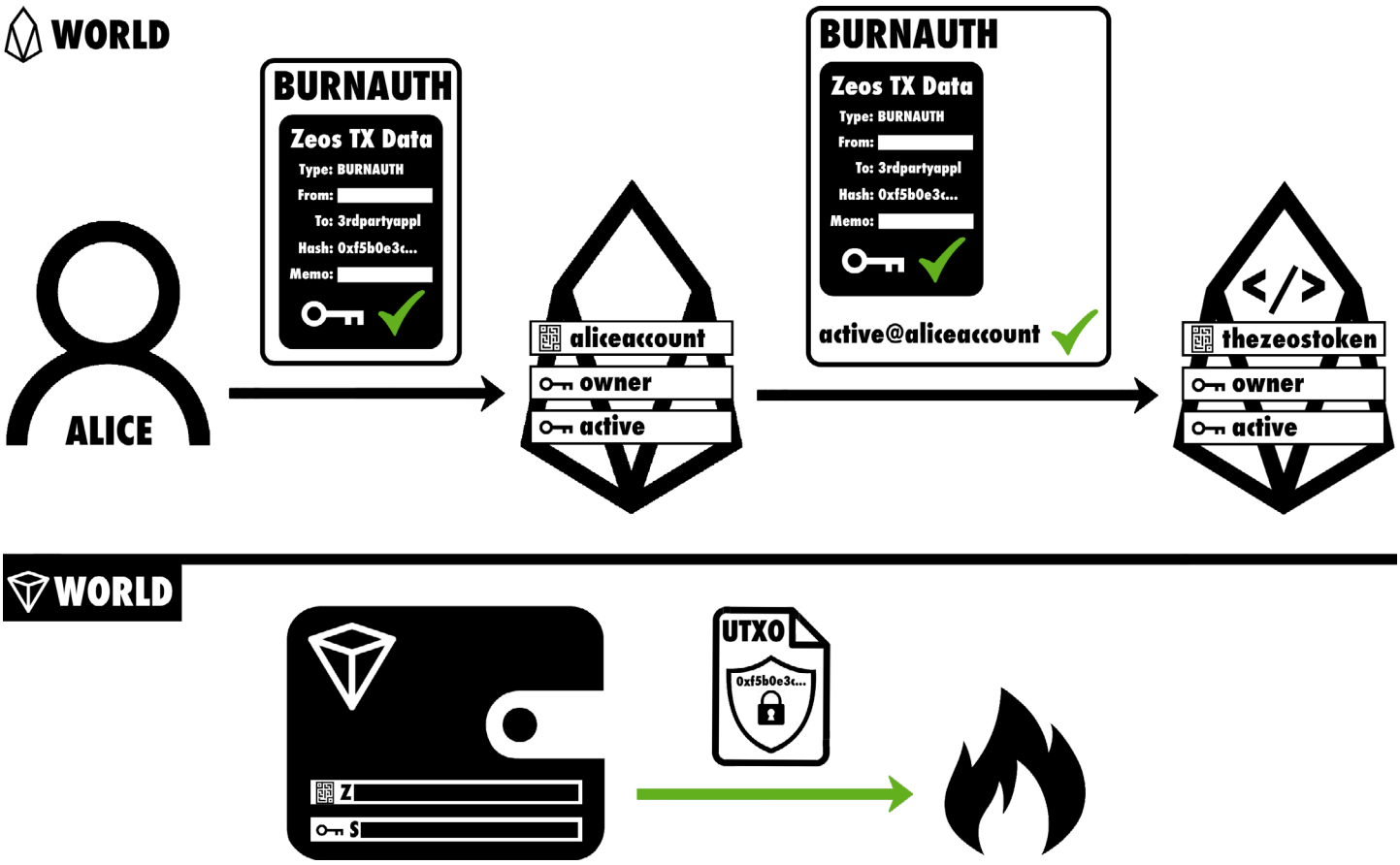
Minting of authentication tokens by calling 'mintauth' is usually done as an inline action by a third-party smart contract which issues the permission. The hash commitment h is stored by the smart contract until the corresponding authentication token is burned by the user – i.e. is used for authentication.



MINTAUTH: ALICE MINTS AN AUTHENTICATION TOKEN AT A PRIVATE ZEOS ADDRESS

BURNING OF AUTHENTICATION UTXOS

In order to gain access to the third-party smart contract the permission – publicly represented by the commitment h and privately represented by the authentication token inside the ZEOS wallet – is burned. To do so a zero knowledge proof is generated that proves knowledge of the secret value *rho* and knowledge of the secret spending key to the *addr* where the authentication token is being held.

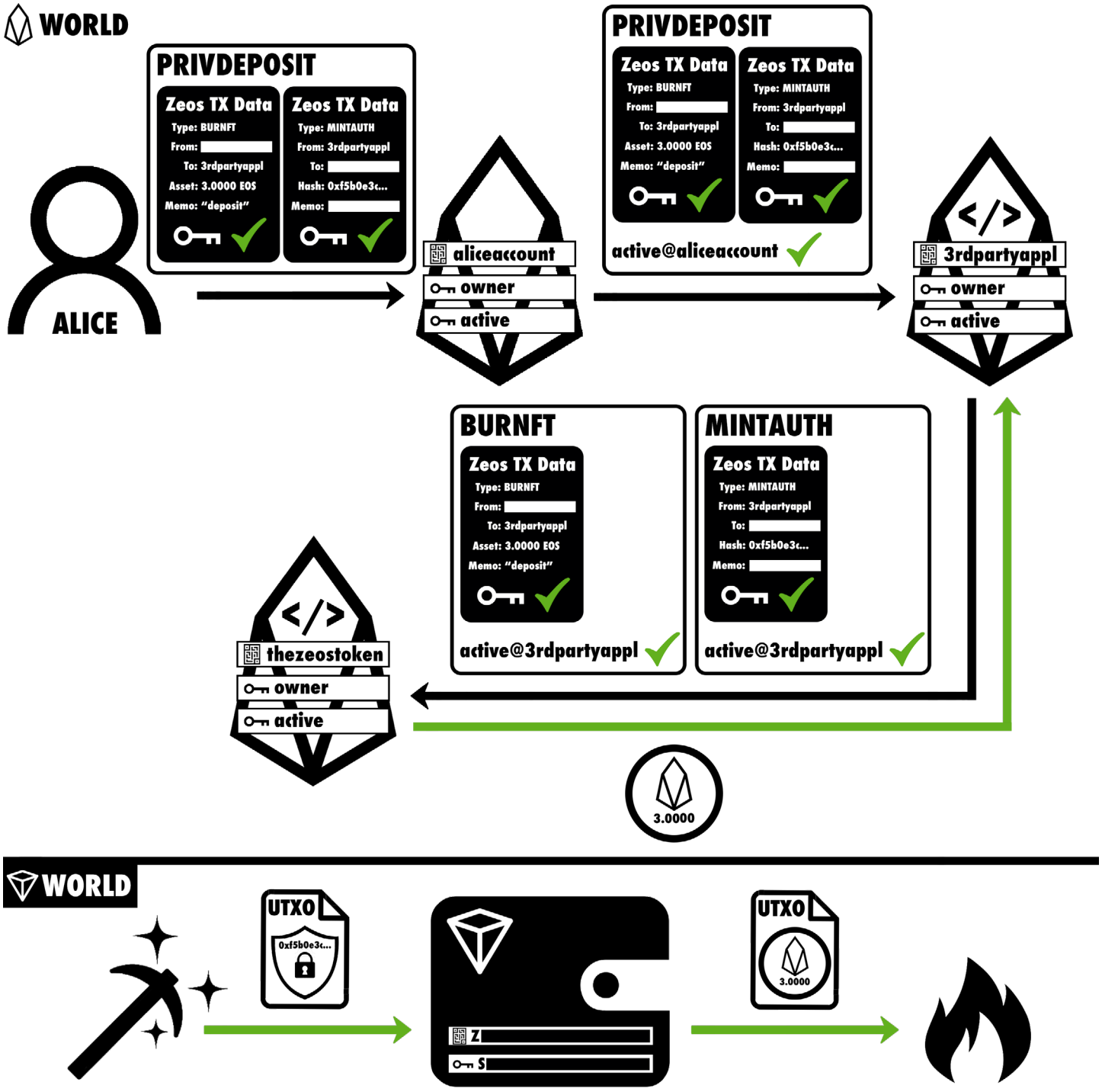


BURNAUTH: ALICE BURNS AN AUTHENTICATION TOKEN AT A PRIVATE ZEOS ADDRESS

PRIVATE DEPOSITS & WITHDRAWALS

Using the mint and burn actions from above, private token deposits and withdrawals can be implemented by *any* third-party smart contract in the EOS ecosystem. This could be achieved by utilizing EOSIO inline actions. For instance, a private deposit action to a custom third-party smart contract could look like this:

```
3RDPARTYAPPL::PRIVDEPOSIT(DPST_PARAMS, AUTH_PARAMS)

    // pseudo code for deposit of fungible tokens. It works analogously
    // for non-fungible tokens: instead of 'quantity' save the NFT's UID.

    // 'burn' the quantity from the user's private wallet into the 3rd party
    //  application's EOS account. NOTE: In your deposit notification handler
    // make sure to ignore incoming transfers from 'thezeostoken'!
    THEZEOSTOKEN::BURNFT(DPST_PARAMS, '3RDPARTYAPPL')

    // 'mint' the user's authentication token
    THEZEOSTOKEN::MINTAUTH(AUTH_PARAMS)

    // save the deposited 'quantity' with the user's 'hash_commitment' as
    // primary key in 'balances' table
    balances[AUTH_PARAMS.hash_commitment] = DPST_PARAMS.quantity;

    // more 3rd party deposit logic...
```



ALICE PRIVATELY DEPOSITS 3 EOS TOKENS FROM A PRIVATE ZEOS WALLET TO A THIRD-PARTY SMART CONTRACT

It embeds two ZEOS inline actions: Firstly the asset to deposit – fungible or non-fungible – is burned from the users private address into the smart contract's EOS account. Secondly, in the same action, the user mints an authentication token and shares the corresponding hash commitment publicly with the smart contract. For traditional deposits and withdrawals EOS smart contracts usually maintain a table where the users' balances are booked under their corresponding EOS account names. In case of private deposits though smart contracts book the balance under the hash commitment the user submits.

USERN (key)	BAL...
geztomzxguge	10...
gi2dmmbzgene	95...
gm3dmnzshege	0.1...
gq3dsnjthage	73...
guydkobvgige	5.3...
...	

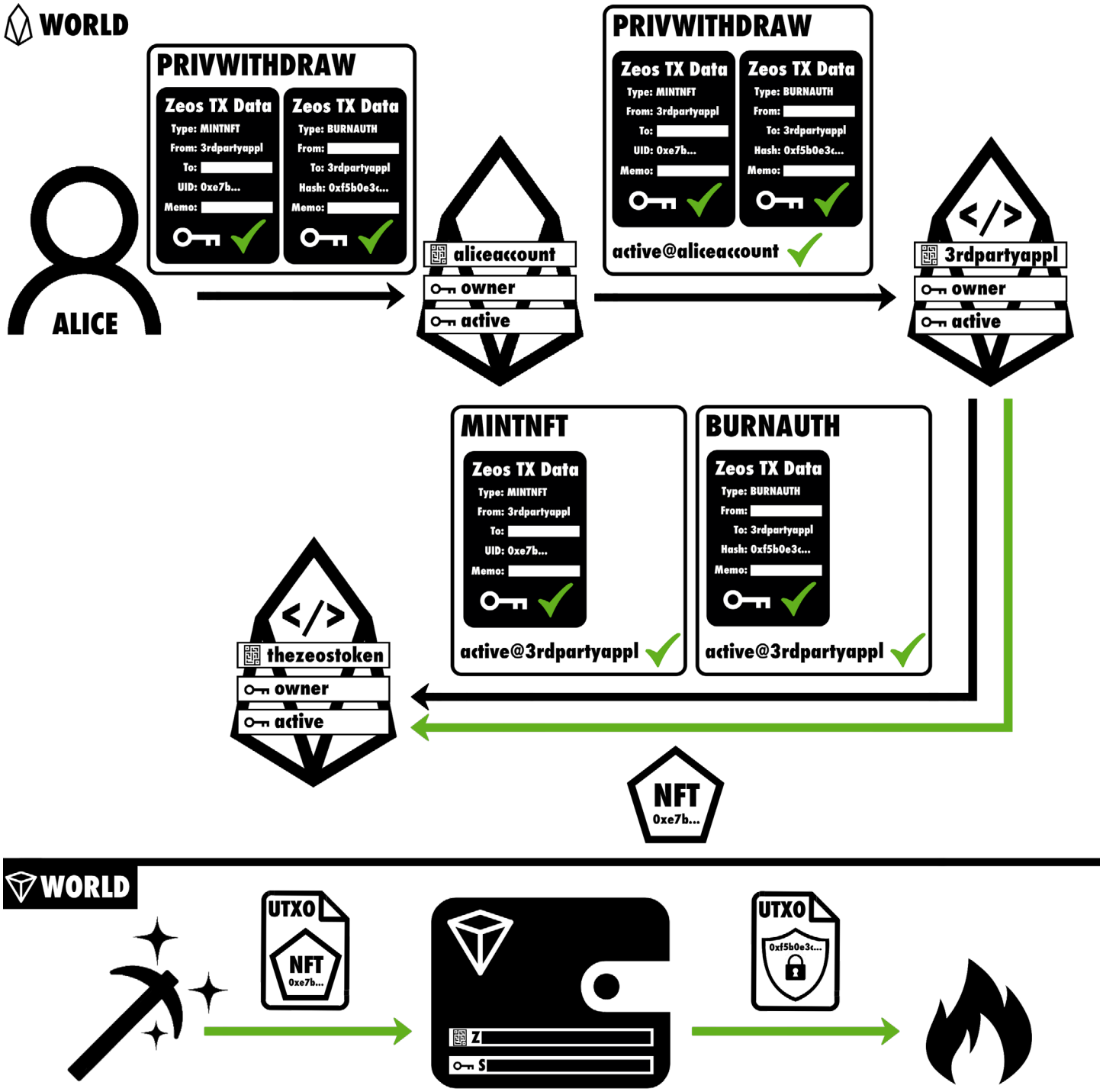
VS

HASH (key)	BAL...
2d0e643f840...	10...
507f8dcdac46...	95...
6cdb5439c8ce...	0.1...
c384782a431...	73...
b0d724bf725...	5.3...
...	

TRADITIONAL EOS BALANCES TABLE VERSUS A ZEOS BALANCES TABLE

A private withdrawal from that smart contract could then be implemented like this:

```
3RDPARTYAPPL::PRIVWITHDRAW(WTHDRW_PARAMS, AUTH_PARAMS)

    // pseudo code for withdrawal of fungible tokens. It works analogously
    // for non-fungible tokens: instead of the uers's balance the NFT with
    // the given UID is minted back into the user's private wallet.

    // 'mint' the quantity from the 3rd party application's EOS account into
    // the user's private wallet.
    THEZEOSTOKEN::MINTFT(WTHDRW_PARAMS, '3RDPARTYAPPL')

    // 'burn' the user's authentication token
    THEZEOSTOKEN::BURNAUTH(AUTH_PARAMS)

    // reset the balance of the user's 'hash_commitment' to zero. Since the
    // authentication token is burned the balance must be withdrawn entirely
    balances[AUTH_PARAMS.hash_commitment] = 0;

    // more 3rd party withdraw logic...
```



ALICE PRIVATELY WITHDRAWS AN NFT FROM A THIRD-PARTY SMART CONTRACT TO A PRIVATE ZEOS WALLET

Like private deposits a private withdrawal embeds two ZEOS inline actions: Firstly the authentication token is burned. By doing so the user authenticates itself by proving that he knows (a) the secret value of rho and (b) the secret spending key to the address of the wallet the authentication token is held at. Only knowledge of those two things can create the public hash commitment under which the balance is stored. Secondly the user mints an UTXO at a private ZEOS address for the asset he withdraws – fungible or non-fungible. By doing so the asset to be withdrawn moves back into custody of the ZEOS token contract while the user is in possession of the corresponding private UTXO.

Based on this concept for private deposits and withdrawals a private 'update' action could be implemented as well. In this case the user burns the permission while at the same time minting a new one for future access. An 'update' action could, of course, contain several ZEOS inline actions such as mints and burns of (non-)fungible assets to eg. partly withdraw or deposit assets to/from a third-party smart contract. The updated balance would be stored under the new hash commitment's value.

Another important thing to note here is that such a private update action could also transfer the permission to another user. This could be done by choosing another wallet address when minting the new authentication token. So while authentication tokens themselves are not transferable, private permissions are indeed transferable by updating them with another wallet address. Needless to say this could not be detected by a public observer since ZEOS addresses are never exposed publicly.

By utilizing ZEOS inline actions all applications on EOS are able to easily integrate private deposits and withdrawals. By exclusively allowing those (i.e. not allowing deposits and withdrawals from transparent EOS accounts) the application would essentially become 'private-by-default'.

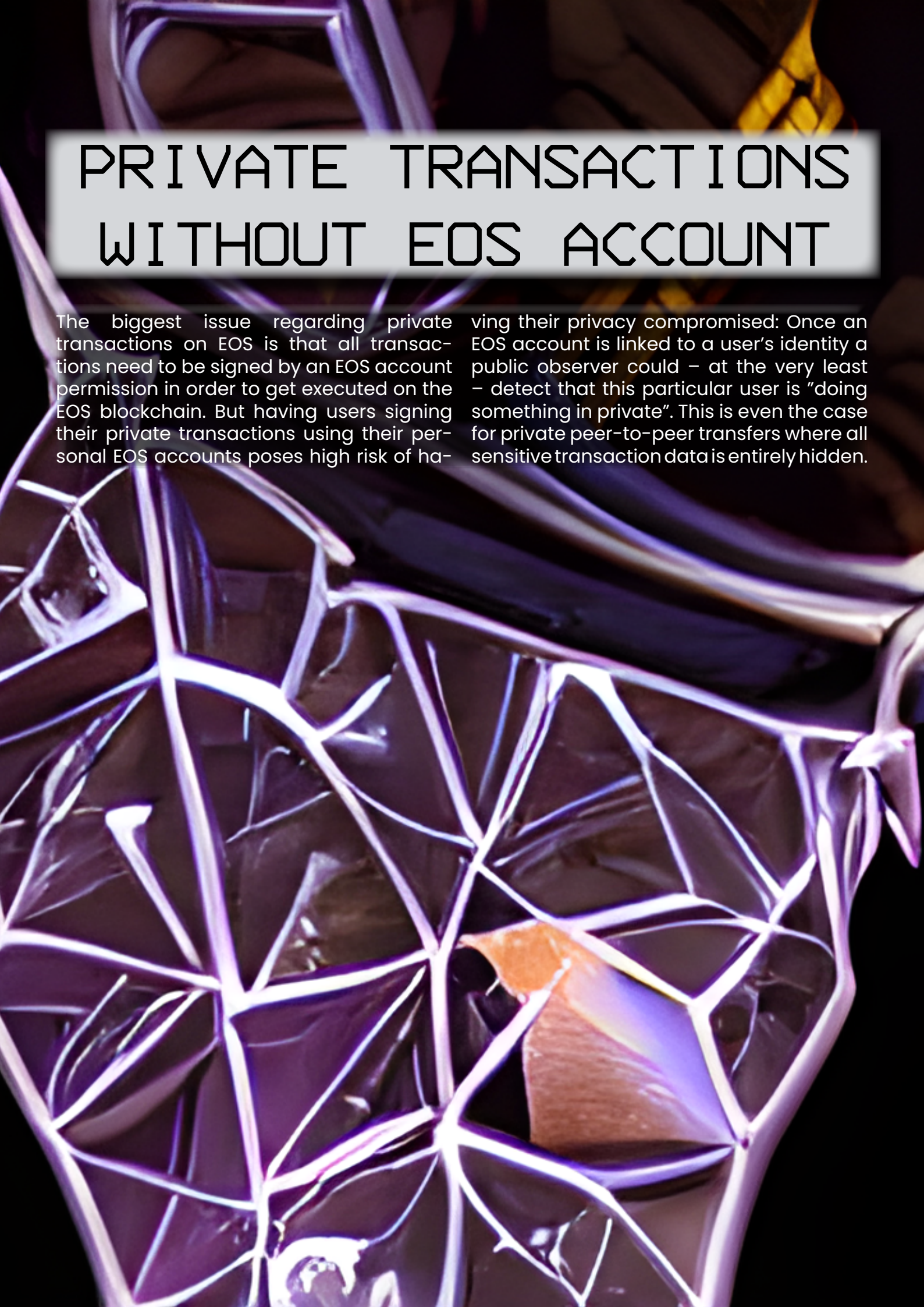
Of course, a hybrid model is possible as well: In this case the smart contract table contains mixed entries of EOS account names and hash commitments as primary keys. The 64-bit EOS account names could be extended to 32 bytes by appending 24 zero bytes. A 64-bit primary key function for those 32-byte values could be implemented by just cutting off the upper 24 bytes returning either the EOS account name values or the lower 64 bits of the hash commitments. Collisions between hash commitments and account names are highly unlikely but need to be handled by the contract.

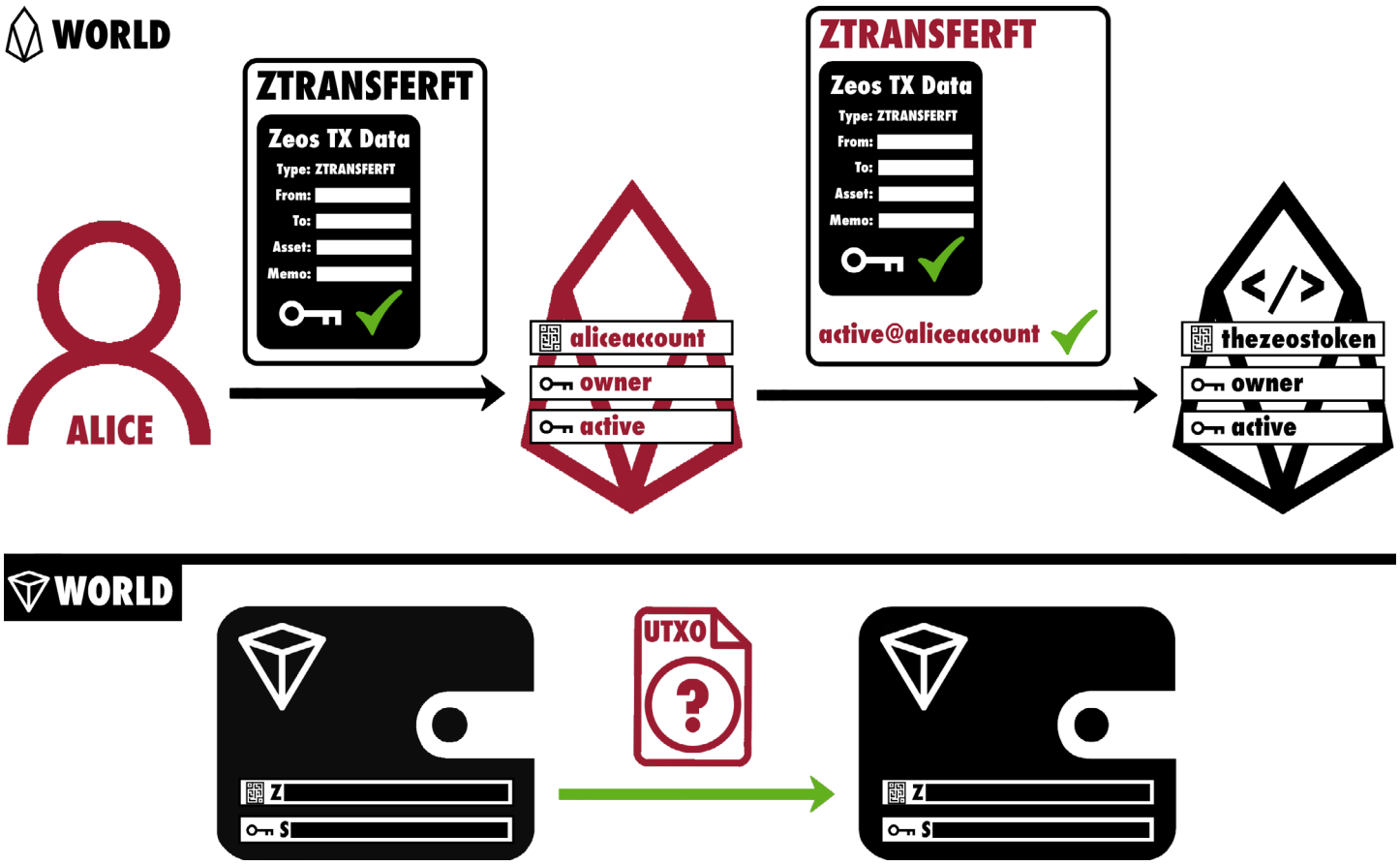
MIXED (key)	BALANCE
000000...000 | geztomzxguge	12.5000 EOS
507f8dcdac46...feb953f1c26ed	255.1822 EOS
c384782a431...87202b8454cc9	10.7500 EOS
000000...000 | gq3dsnjthage	995.1000 EOS
b0d724bf725...02f6d1af9a42c	0.8912 EOS
...	...

Note: In case an application adapts private deposits/withdrawals the notification handler used for token deposits of that contract has to handle incoming transfers from 'thezeostoken' contract separately – i.e. ignore them since those balances are already handled by the private deposit and withdraw actions.

PRIVATE TRANSACTIONS WITHOUT EOS ACCOUNT

The biggest issue regarding private transactions on EOS is that all transactions need to be signed by an EOS account permission in order to get executed on the EOS blockchain. But having users signing their private transactions using their personal EOS accounts poses high risk of having their privacy compromised: Once an EOS account is linked to a user's identity a public observer could – at the very least – detect that this particular user is "doing something in private". This is even the case for private peer-to-peer transfers where all sensitive transaction data is entirely hidden.





IF ALICE'S EOS ACCOUNT IS LINKED TO HER IDENTITY PRIVATE TRANSACTIONS SIGNED WITH HER ACCOUNT CAN BE LINKED TO ALICE COMPROMISING (AT LEAST PART OF) HER PRIVACY

For private peer-to-contract transactions the situation looks even worse: When depositing or withdrawing privately to or from a third-party smart contract all information about which assets are being transferred is already publicly exposed. The only thing remaining private in such a transaction is the user's wallet address. This is usually enough to keep the user's identity anonymous. However, since the user's EOS account which is being used to sign the private transaction is already linked to his identity an observer could detect that "user X deposited asset Y to smart contract Z" which basically means no privacy at all.

This, of course, is a big issue which could be solved by requiring users to *anonymously* create a second EOS account which would then be used *exclusively* for signing private transactions. In this approach users would use their publicly known EOS account only to move assets in and out of their private ZEOS wallets. All other private transactions would be signed only with their second, anonymous EOS account.

That way users achieve full privacy as long as their second EOS account *never* gets linked to their identity. While this approach provides a solution for strong privacy on EOS, it requires everyone to manage two accounts – including resources – which is not a very user-friendly solution. Furthermore, executing just one single transaction using the wrong account could potentially compromise the owners privacy entirely – including all past transactions.

Furthermore users could *unknowingly* create links between their accounts simply by accident through resource management like having one account powering up the other or vice versa. There are just

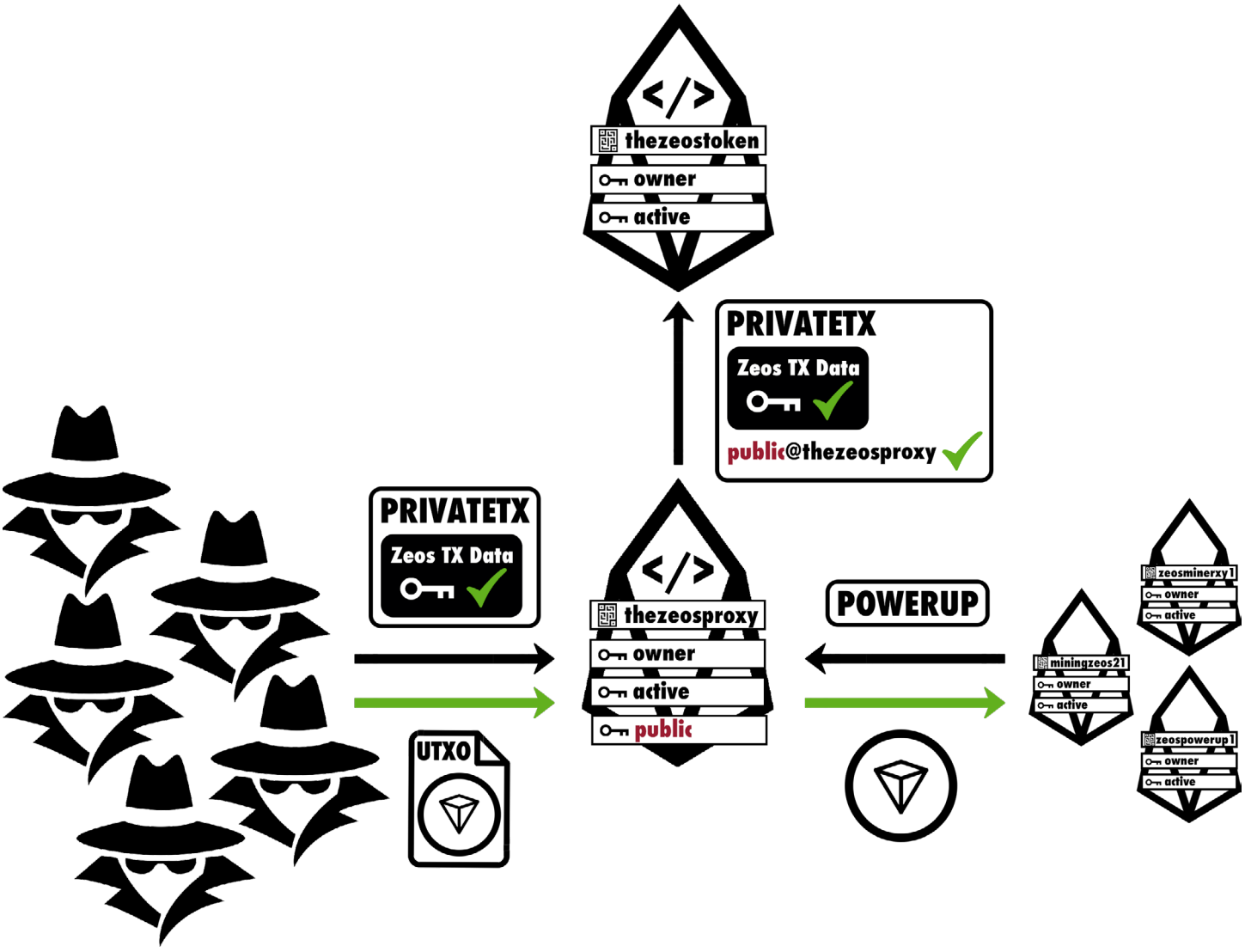
too many ways users could screw up their privacy without even noticing.

To achieve true and fool-proof privacy on EOS it is crucial to un-link private transactions from users' EOS accounts.

While maintenance of two EOS accounts could theoretically be managed intelligently by a wallet software it still poses the risk of human error. Ideally, users shouldn't even be required to have two EOS accounts at all nor have to worry about which one to use for what kind of transaction.

This is why ZEOS introduces a transaction fee model enabling private ZEOS transactions without an EOS account. This is possible thanks to the EOSIO permission system and the 'Contract Pays' feature which is going to be introduced in the upcoming 'Mandel' fork of the EOSIO codebase. In this model, all users sign private ZEOS transactions using one and the same EOSIO permission of a so-called 'ZEOS Proxy Account'.

The proxy account has a special 'public' permission whose private key is public and can be used by anyone to sign private ZEOS transactions. Instead of paying for the necessary EOS resources directly when executing a private transaction, a fee denominated in ZEOS is paid *privately* by the transacting user to the proxy's EOS account. This is done effectively by just 'burning' a UTXO from the user's private ZEOS wallet right into the proxy's EOS account. So called 'miners' on the other hand earn those transaction fees by powering up the proxy account's CPU, NET and RAM resources.

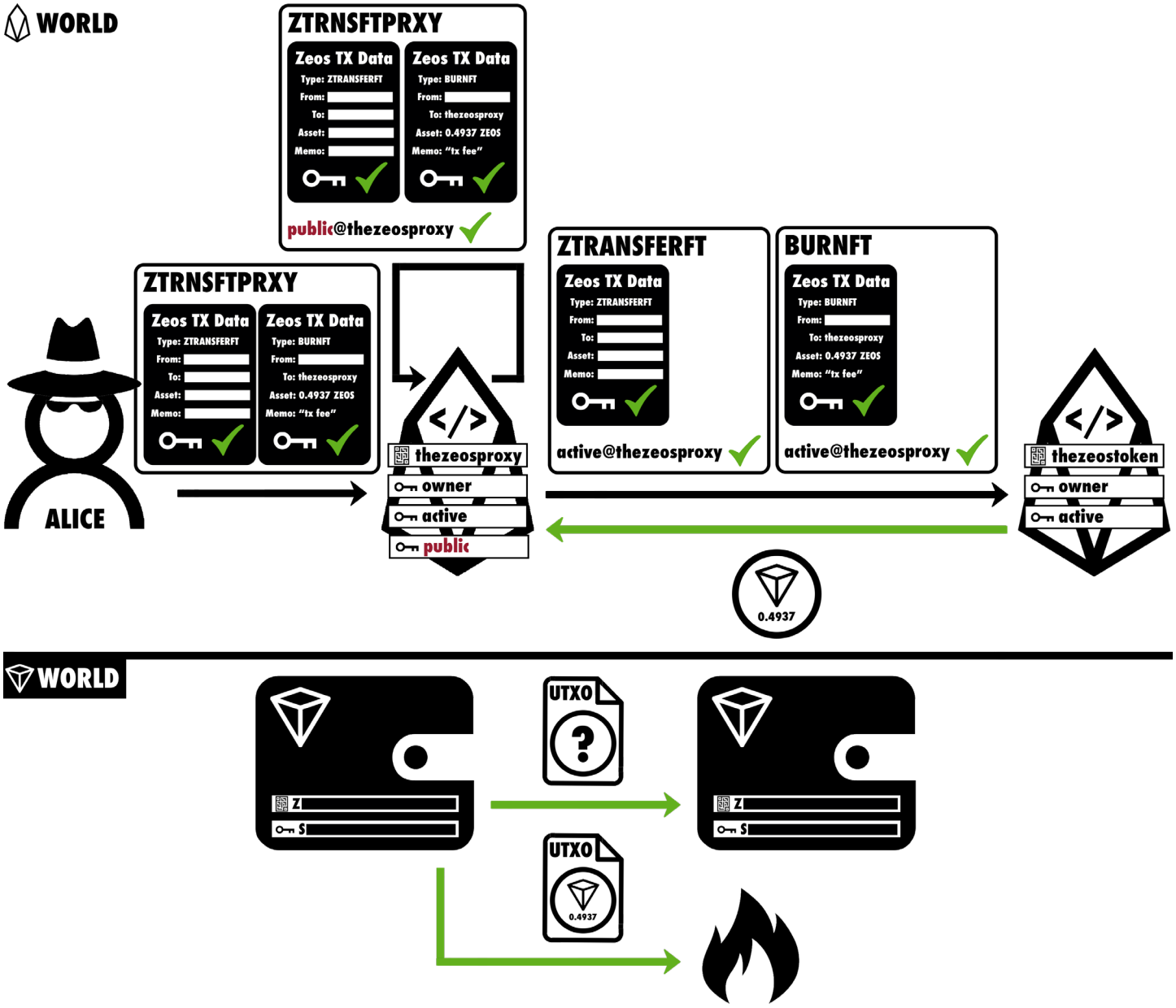


EVERYONE USES THE 'PUBLIC' PERMISSION TO ANONYMOUSLY SIGN PRIVATE TRANSACTIONS BY PAYING FEES DENOMINATED IN ZEOS WHICH 'MINERS' COMPETE FOR

THE PROXY

The proxy account has a special 'public' permission whose private key is publicly shared. This means anyone can use this permission to sign EOS transactions. However, the permission only allows for execution of specific actions of the proxy account's smart contract. Those proxy actions are nothing but wrapped actions for ZEOS private transactions. Such a wrapped action includes the actual private transaction to be executed plus an inline call to 'burn' a UTXO from the user's private wallet into the proxy account effectively paying the transaction fee. Executing a private peer-to-peer 'ztransfer' in the fee model would look like this:

```
THEZEOSPROXY::ZTRNSFTPRXY(ZTRANSFER_PARAMS, FEE_PARAMS)

    // pseudo code for private transfer of fungible tokens. It works analogously
    // for non-fungible tokens. The transaction fee is burned from the users
    // wallet into the proxy's EOS account. If the fee is not high enough to cover
    // all EOS resources this action fails.

    // Execute the ztransfer
    THEZEOSTOKEN::ZTRANSFERFT(ZTRANSFER_PARAMS)

    // 'burn' the transaction fee into the proxy account
    THEZEOSTOKEN::BURNFT(FEE_PARAMS, 'THEZEOSPROXY')
```



BY PRIVATELY TRANSACTING USING THE ZEOS PROXY ALICE REMAINS FULLY ANONYMOUS

Since the burn action reveals the asset's amount and type the correct transaction fee can easily be enforced by the proxy's wrapped action. Only if the fee denominated in ZEOS is high enough to cover the EOS resources of the transaction (plus a small premium as incentive for ZEOS miners) does it gets executed by the proxy account. The resources that need to be paid for are CPU, NET and potentially RAM (in the event of a private deposit to a third-party smart contract). Fees always have to be paid in ZEOS.

No other token is accepted by the proxy account so as not to reveal any information about the assets being transferred.

By having all EOS resources – CPU, NET and RAM – dynamically being paid for in ZEOS makes ZEOS a 'gas token' which powers private access to the EOS economy enabled by the ZEOS proxy account. The 'gas token' character makes ZEOS comparable to ETH which is Ethereum's gas token.

THE MINER

Anyone who wants to earn ZEOS tokens by providing EOS resources to the proxy account can become a 'miner'. A miner permanently monitors the proxy account's available resources. If the remaining CPU/NET resources fall below a certain threshold miners are able to execute a wrapped action of the proxy account to power it up. The wrapped action includes two inline actions: One is the actual call to power-up the resources. The other one is a transfer to payout the accumulated transaction fees in the form of ZEOS tokens to the miner's EOS account. So only if the power-up gets executed does the miner receive the reward.

The proxy account disposes of a second power-up action for RAM. If RAM is consumed by private deposits an additional fee will be paid by the user to cover the RAM cost. Therefore a second balance is maintained by the ZEOS Proxy only for the accumulation of RAM fees. For RAM management the exact same competition among miners takes place: If the remaining RAM of the proxy account falls below a certain threshold miners are able to execute the second power-up action which purchases RAM[3] for the proxy account and pays out the accumulated RAM fees – denominated in ZEOS – to the miner's EOS account.

Since the power-up actions of the proxy are only executable under certain conditions it adds a little game theory to the mining process: Miners are competing for the fee pots and they can only claim them if they are the ones executing the first power-up below the threshold. Normal power-ups could be used by anyone to interfere in the competition increasing the fee pot at no cost for the miners. Miners could also execute private transactions themselves to make the pot(s) claimable before the next random user does. In any case: miners only pay for the resources if they are also the ones receiving the pot. The pot reward on the other hand should always cover the costs for the power-up or RAM purchase, respectively – plus a little extra for the miner.

In addition to the miner premium (which is set through ZEOS governance) a second premium could be added to the transaction fees. This would open up a way for the protocol to generate an income and fund the work of the ZEOS smart organization which is governing the ZEOS ecosystem.

3 Miners would have to maintain a small EOS balance on the proxy account in order to be able to execute power-ups or RAM purchases via inline action calls. For the sake of simplicity this is being ignored in this paper.

CONCLUSION

The ZEOS fee model based on the concept of only one public account permission signing all private ZEOS transactions is what makes the protocol truly private and fool-proof. The only interactions with the protocol using a personal, link-able EOS account is when assets are being moved in or out of private ZEOS wallets. All other transactions like private token transfers or private interactions with third-party smart contracts are being paid for with transaction fees denominated in ZEOS and are signed by the proxy account's public permission.

This has even much broader implications: ZEOS wallets become usable for anyone outside the EOS ecosystem since no personal EOS account is required to transact privately. This would allow anyone outside of EOS to hold all kinds of EOS assets in full privacy using only ZEOS wallets.

Furthermore it enables those users to privately interact with all kinds of EOS applications that choose to integrate ZEOS private deposits and withdrawals. All that is needed is a ZEOS wallet which can be created at the click of a button and, of course, some ZEOS tokens to cover the transaction fees.

Additionally, traditional web applications like centralized exchanges could very easily integrate ZEOS wallets as well since they work almost exactly the same way as traditional Bitcoin wallets. For instance: Centralized exchanges could list any EOS asset and allow for deposits and withdrawals to the exchange using private ZEOS addresses instead of traditional EOS accounts. This would make all user deposits and withdrawals of – no matter what – EOS asset 'private-by-default' .

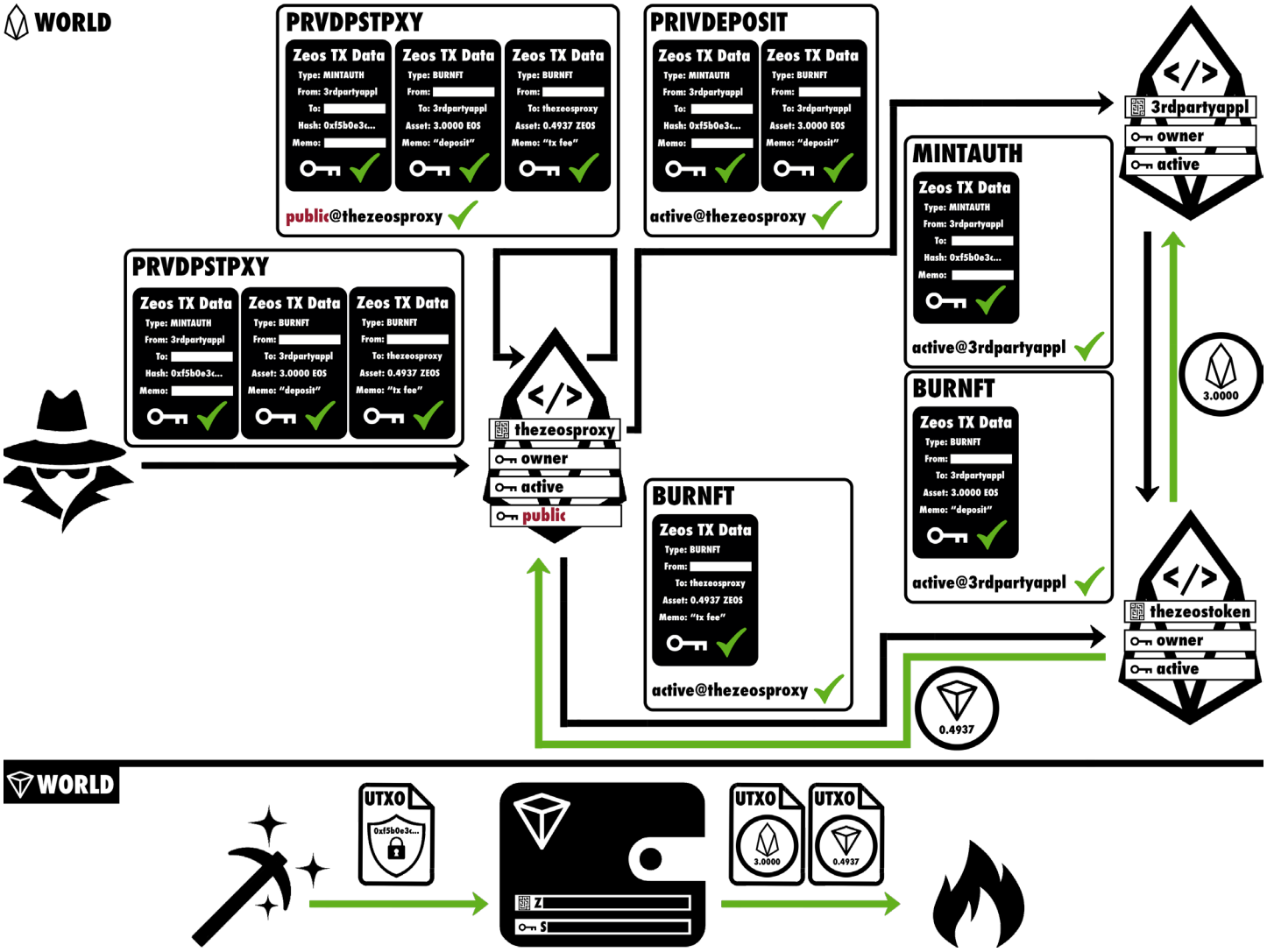
The usage of ZEOS wallets instead of transparent EOS accounts makes *any* asset on EOS an arguably stronger privacy coin than Monero. ZEOS offers fully private token transfers plus potentially private access to any DeFi application built on the very same blockchain – something that native privacy coins like Monero are technologically incapable of due to their lack of smart contract functionality. ZEOS private deposits and withdrawals without EOS account could be a game changer not only for EOS but for the entire DeFi industry.

PRIVATE DEPOSIT & WITHDRAWAL USING THE PROXY ACCOUNT

For completion the most complex use cases of the ZEOS proxy account are depicted here as well: Private deposit and withdrawal to/from a third-party smart contract.

A private deposit of fungible tokens could be implemented like this (NFT analogous):

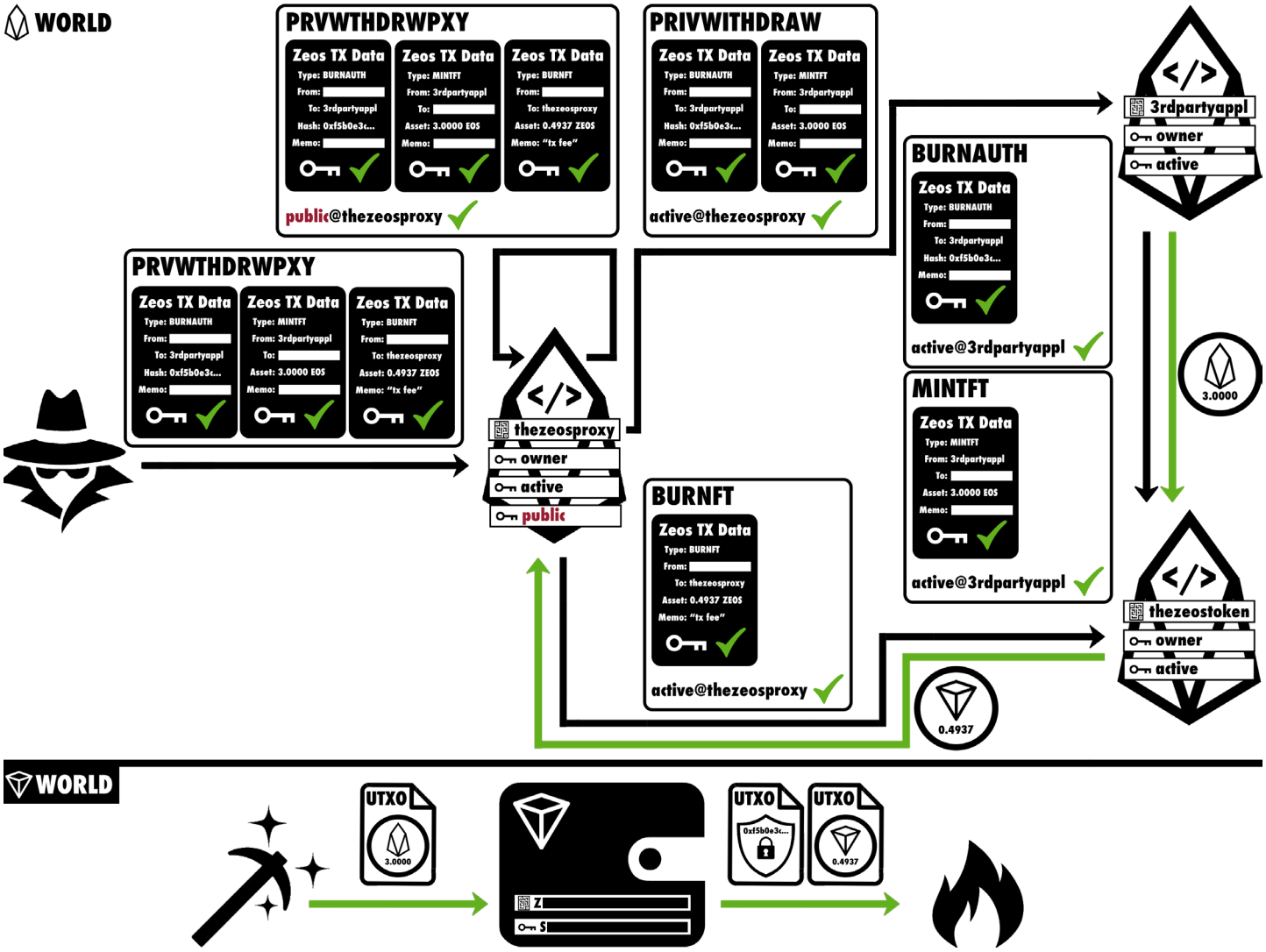
```
THEZEOSPROXY::PRVDPSTPXY(FEE_PARAMS, DPST_PARAMS, AUTH_PARAMS)

    3RDPARTYAPPL::PRIVDEPOSIT(DPST_PARAMS, AUTH_PARAMS)

        // 'burn' the quantity from the user's private wallet into the 3rd party
        //  application's EOS account. NOTE: In your deposit notification handler
        // make sure to ignore incoming transfers from 'thezeostoken'!
        THEZEOSTOKEN::BURNFT(DPST_PARAMS, '3RDPARTYAPPL')

        // 'mint' the user's authentication token
        THEZEOSTOKEN::MINTAUTH(AUTH_PARAMS)

        // 3rd party deposit logic...

    // 'burn' the transaction fee into the proxy account
    THEZEOSTOKEN::BURNFT(FEE_PARAMS, 'THEZEOSPROXY')
```



A USER PRIVATELY DEPOSITS 3 EOS TOKENS FROM HIS ZEOS WALLET TO A THIRD-PARTY SMART CONTRACT USING THE PROXY ACCOUNT BY PAYING A TRANSACTION FEE DENOMINATED IN ZEOS

A private withdrawal of fungible tokens could be implemented like this (NFT analogous):

```
THEZEOSPROXY::PRVWTHDRWPXY(FEE_PARAMS, WTHDRW_PARAMS, AUTH_PARAMS)

    3RDPARTYAPPL::PRIVWITHDRAW(WTHDRW_PARAMS, AUTH_PARAMS)

        // 'mint' the quantity from the 3rd party application's EOS account into
        // the user's private wallet.
        THEZEOSTOKEN::MINTFT(WTHDRW_PARAMS, '3RDPARTYAPPL')

        // 'burn' the user's authentication token
        THEZEOSTOKEN::BURNAUTH(AUTH_PARAMS)

        // 3rd party withdraw logic...

    // 'burn' the transaction fee into the proxy account
    THEZEOSTOKEN::BURNFT(FEE_PARAMS, 'THEZEOSPROXY')
```



A USER PRIVATELY WITHDRAWS 3 EOS TOKENS FROM A THIRD-PARTY SMART CONTRACT INTO HIS ZEOS WALLET USING THE PROXY ACCOUNT BY PAYING A TRANSACTION FEE DENOMINATED IN ZEOS

ZASSETS

Inspired by the Haven Protocol zAssets are synthetic assets which represent any kind of real world asset such as fiat currencies, precious metals, commodities or stocks. All that is required to create a zAsset is a price oracle for the corresponding real world asset. zAssets are minted by users in return for burning an equivalent amount of ZEOS tokens and/or by locking up external collateral in the protocol. In contrast to Haven though, the market capitalization of all circulating zAssets in existence is auditable since the mint and burn process of zAssets is fully transparent.

The transparency of zAssets and therefore knowledge about the exact valuation of all synthetic assets in existence allows for protective mechanisms to be built into the protocol. For instance the protocol should slow down and eventually completely stop the burning of ZEOS tokens to create synthetic assets in order to protect it from becoming over-leveraged. Something that most of the so-called 'algorithmic stablecoins' in the crypto space lack. Instead of exclusively burning ZEOS tokens with no limits, the protocol automatically starts to slow down the ZEOS burn-rate and instead locks external collateral if the demand for synthetic assets outpaces the growth of the ZEOS market capitalization.

All zAssets are standard fungible EOS tokens issued by the ZEOS token contract. Like all other assets on EOS they can move freely between private ZEOS wallets and transparent EOS accounts. This means they can be listed on all EOS exchanges and integrated into any application on EOS – just like any other EOS asset. This adds a whole new class of assets to the DeFi ecosystem on EOS.

Unlike many other DeFi protocols for synthetic stable assets in the crypto space, zAssets are not purely algorithmic. This approach is inspired by the Frax Protocol but with a difference: Instead of only requiring ZEOS tokens to be burned the protocol maintains a pool of external collateral which gradually increases depending on how *leveraged* the protocol is. The protocol leverage ratio L is defined as:

$$L \overset{\text{def}}{=} \frac{M_{zeos}}{M_{syn}}$$

with:

$$M_{zeos} = s_{zeos} \cdot p_{zeos}$$

$$M_{syn} = \sum_{asset \in zAssets} s_{asset} \cdot p_{asset}$$

where:
s_x is the supply of asset x
p_x is the price of asset x

The protocol leverage is defined as the ratio of the market capitalization of the ZEOS token M_{zeos} itself to the market capitalization of all circulating synthetic assets M_{syn} in existence. The leverage ratio L determines how many times all circulating zAssets are backed by the actual valuation of the ZEOS token. If this leverage ratio reaches a certain threshold the protocol starts to accumulate external collateral since the demand for zAssets outpaces the growth of the ZEOS market capitalization.

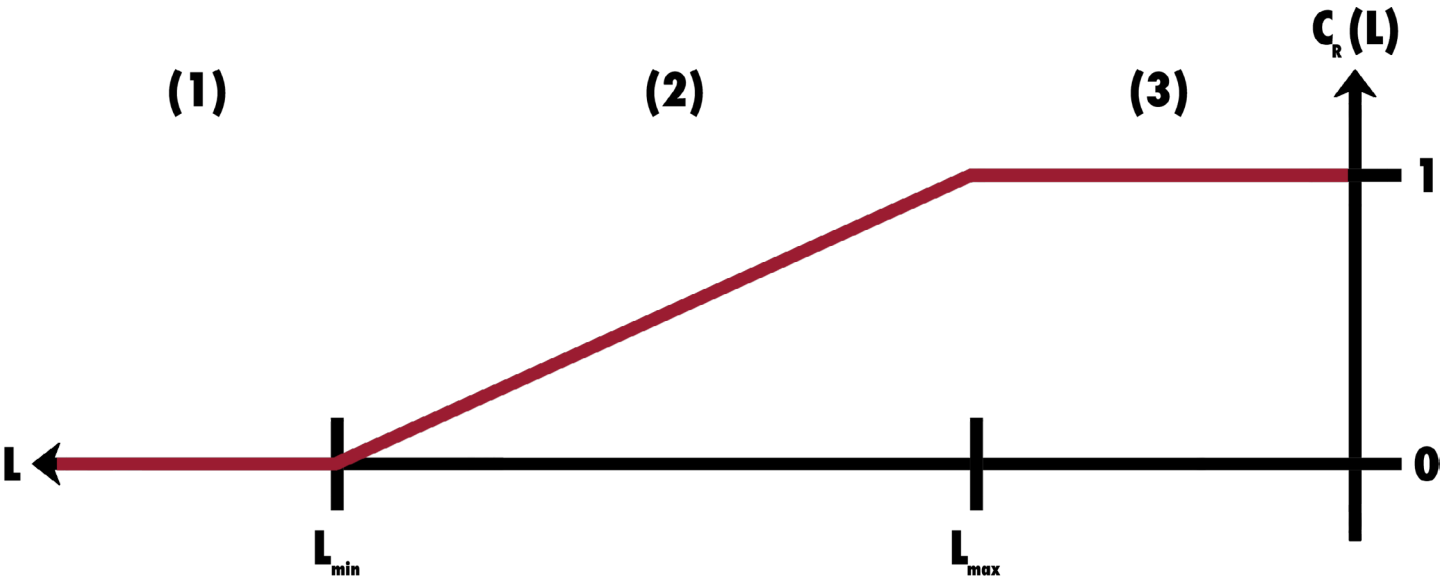
Depending on the leverage ratio L the external collateral C_{ext} which is required by the protocol is determined by:

$$C_{ext} \overset{!}{=} C_R(L) \cdot M_{syn}$$

Where $C_R(L)$ is the collateral ratio of the protocol defined as a function of L:

$$C_R(L) = \begin{cases} 0 & if \quad L_{min} < L \quad (1) \\ \frac{L - L_{min}}{L_{max} - L_{min}} & if \; L_{max} < L < L_{min} \quad (2) \\ 1 & if \quad L < L_{max} \quad (3) \end{cases}$$

Where L_{min} and L_{max} are threshold values set by protocol governance. L_{min} determines when the protocol starts to lock external capital and L_{max} determines when the protocol exclusively requires external collateral in order to mint synthetic assets. Together both values implicitly determine the burn- and mint-rate of ZEOS as collateral for zAssets.



A SMALLER LEVERAGE RATIO L RESULTS IN A HIGHER COLLATERAL RATIO CR EFFECTIVELY INCREASING THE REQUIREMENT OF EXTERNAL COLLATERAL TO BE LOCKED

Hence, the protocol has three different states which seamlessly transition into each other: (1) purely algorithmic/non-collateralized, (2) fractionally algorithmic/collateralized and (3) non-algorithmic/fully collateralized.

(1) Purely Algorithmic

If the leverage ratio L is greater than the minimum ratio, L_{min}, the protocol is in the purely algorithmic state where only ZEOS tokens are being burnt to mint zAssets and vice versa. In this state no external collateral is accumulated by the protocol.

(2) Fractionally Algorithmic/Collateralized

If the leverage ratio L is between L_{min} and a maximum leverage ratio L_{max} the protocol is in a fractionally algorithmic state. In this state the requirement of external collateral C_{ext} equals a certain percentage of the total market capitalization of all circulating zAssets M_{syn}. To mint new synthetic assets users will have to partly add some external collateral and partly burn ZEOS tokens. Burning synthetic assets on the other hand will return some external collateral to the user as well as mint some ZEOS tokens back into existence. Only if the external colla-

teral value C_{ext} becomes too small will the protocol (mostly or exclusively) mint ZEOS tokens back instead of returning external collateral when burning zAssets. Alternatively if the valuation of external collateral is above the required level the protocol will (mostly or exclusively) return external collateral to the user instead of minting ZEOS tokens back.

(3) Fully Collateralized

If the leverage ratio falls below L_{max} the protocol enters the fully collateralized state. In this state no ZEOS tokens are burned anymore. All zAssets minted in this state require full backing by external collateral. Burning synthetic assets will return external collateral to the users instead of minting new ZEOS tokens. Only if the overall valuation of external collateral becomes too low the protocol will start minting ZEOS tokens again in order to increase the true external collateral ratio and to be able to pay the users in return for burning zAssets.

With each and every user interaction – i.e. minting and burning zAssets – the protocol tries to maintain the correct amount of external collateral C_{ext} determined by the leverage ratio L. Since the values of M_{syn}, M_{zeos} but also C_{ext} itself are constantly changing due to price fluctuations on the open markets, the requirement of external collateral C_{ext} changes permanently as well. While users always get the exact same value from the protocol as they add to it, the ratio of ZEOS tokens being burned (or minted) versus external collateral to be added (or removed) might change with each and every transaction.

Balancing the correct amount of external collateral C_{ext} as determined by the current protocol leverage L is the protocols only priority.

COLLATERAL

The external collateral C_{ext} locked inside the protocol consists of two assets:

- The native EOS token
- The Vigor stable token which is pegged to the US Dollar

By choosing one crypto asset with a significantly higher market capitalization than the ZEOS token itself – EOS – plus one stable token which is pegged to the Dollar – Vigor –, the overall valuation of the external collateral C_{ext} is expected to be much less volatile than the ZEOS market capitalization M_{zeos} itself.

The value of the total external collateral C_{ext} is the sum of the total value of EOS collateral plus the total value of the Vigor collateral:

$$C_{ext} = C_{eos} + C_{vigor}$$

The protocol always tries to balance the two collateral pools: The value of all EOS tokens locked inside the protocol should always equal the value of all Vigor stable tokens locked. Both fluctuate in value and therefore the ratio in which users have to add or remove EOS and Vigor tokens when minting or burning zAssets fluctuates as well.

$$C_{eos} \stackrel{!}{=} C_{vigor}$$

The condition, of always keeping the two in balance results in the following accumulation behavior: In a crypto bull market when minting zAssets the protocol requires nominally more Vigor tokens to be added than EOS since EOS is rising in value against Vigor. When burning zAssets, users would receive more EOS tokens back than Vigor.

In a bear market, on the other hand, it would be the opposite: The protocol instead accumulates EOS tokens while paying out predominantly Vigor. This anti-cyclic accumulation behavior allows users to get rid of their 'weak' assets when minting zAssets and to receive 'strong' assets when burning zAssets – depending on the respective market phase.

MINTING & BURNING OF ZASSETS

Depending on the current state of the protocol as well as the valuation of the external collateral C_{ext} currently locked inside the protocol, minting and burning of zAssets will require a different ratio of ZEOS, EOS and Vigor tokens to be added or removed, respectively.

For instance, a user wants to mint (or burn) a certain amount a_X of synthetic asset X. This asset has a price p_x on the open market defined by a corresponding price oracle. The following equation applies:

$$a_X \cdot p_X = a_{zeos} \cdot p_{zeos} + a_{eos} \cdot p_{eos} + a_{vigor} \cdot p_{vigor}$$

The amounts a_{zeos}, a_{eos} and a_{vigor} are now to be determined such that the protocol's actual value of external collateral C_{ext} approximates the value of external collateral required – including the current transaction – best possibly.

No matter if minting or burning zAssets and no matter at what time: Users always receive the *exact same* value *from* the protocol as they put *into* the protocol – measured at the time of the transaction. Only the *ratio* of ZEOS, EOS and Vigor changes according to their current valuations and the current state of the protocol.

RE-BALANCING COLLATERAL

Under certain market conditions the requirement for external collateral might rise much quicker than the protocol is able to 'catch up' with only through ongoing minting and burning of zAssets by the users. That's why a 're-balance' action is introduced which may be executed by users in case the value of collateral required deviates *too far* from the actual value of collateral locked inside the protocol. The percentage by which the collateral value may deviate before the re-balance action becomes executable is set through governance.

If the protocol becomes too under-collateralized users are able to execute the re-balance action in order to add more EOS and Vigor tokens to the protocol in return for newly minted ZEOS tokens. This results in more external collateral getting locked inside the protocol as well as an increase in market capitalization of ZEOS since more tokens get into circulation. Both adds to the solvency of the protocol.

If the protocol becomes too over-collateralized on the other hand, users are able to execute the re-balance action to effectively trade ZEOS tokens for external collateral from the protocol. The ZEOS tokens added are getting burned by the protocol while an equivalent amount of external collateral is released to pay the user. This results in a decrease of ZEOS market capitalization as well as a decrease in collateral being locked. Both reduces the solvency of the protocol while increasing the scarcity of ZEOS.

TIME DELAYS & MOVING AVERAGES

To prevent abuse of the protocol's mint and burn tokenomics during times of stressed markets or through short-term oracle price manipulations two measures are introduced to filter out high-frequency price fluctuations:

- time delays on the ability to burn zAssets
- moving averages for oracle price values

A time delay prevents users from burning zAssets right after having them minted. This makes the protocol more attractive to long-term traders and investors instead of short-term traders who could take advantage of the tokenomics: Since minting and burning of zAssets involves only small trading fees but basically provides infinite liquidity, traders could cause small fluctuations of the ZEOS token price themselves while using the price differences to mint and burn zAssets back and forth and hence accumulate ZEOS tokens at the cost of inflation. A simple time delay on the ability to burn zAssets solves this issue.

Furthermore, moving averages are used for all oracle values to filter out high-frequency fluctuations. This smoothes out all price signals going into the protocol which leads to a less oscillating and hence more stable protocol state. In addition to that, it is also a measure against short-term oracle price manipulations.

The time delay value and the window size of the moving averages are set through protocol governance.

CAPITAL EFFICIENCY

The elastic token supply of ZEOS allows for zAssets to be way more capital efficient than over-collateralized debt positions (CDPs) like Vigor and other stable assets of that category. For instance: Several Dollars worth of collateral need to be locked up to create only one Dollar worth of CDP. In addition to this interest needs to be paid for any Dollar of CDP in existence.

This makes CDPs very capital in-efficient but also relatively low-risk. In contrast the ZEOS protocol for zAssets is more capital efficient since only one Dollar worth of ZEOS and/or external collateral is required to create one Dollar worth of zAssets. Needless to say, this comes at the cost of risk for ZEOS token holders.

While only the market capitalization of the ZEOS token itself, M_{zeos}, in relation to the valuation of all zAssets, M_{syn}, is used to determine the protocol leverage ratio L, the protocol is actually backed by both: The market capitalization of ZEOS as well as the external collateral C_{ext} locked inside the protocol. Therefore the *true* leverage ratio which determines the overall solvency S of the protocol is defined by:

$$S = L_{true} \stackrel{\text{def}}{=} \frac{M_{zeos} + C_{ext}}{M_{syn}}$$

The value of S must never fall below 1.0 in order for the protocol not to get liquidated.

It is up to the protocol governance to set the boundaries L_{min} and L_{max} responsibly high enough so that the protocol is able to survive black swan events. It is crucial that the protocol transitions early enough from a purely algorithmic state into the fractionally collateralized state and hence starts to lock external collateral. In addition the protocol should as well transition into the fully collateralized state early enough in order to have enough 'buffer' for downside risk.

L_{min} and L_{max} implicitly determine the burn-rate of ZEOS tokens: The higher the value of L_{min}, for instance, the earlier the protocol starts to accumulate external collateral which effectively means lowering the burn-rate of ZEOS tokens. The higher the value of L_{max} the earlier the protocol stops burning ZEOS tokens entirely and exclusively requires external collateral to be locked instead. Lowering both values increases the burn-rate of ZEOS but also results in a higher risk for ZEOS token holders.

The solvency of the protocol largely depends on the market capitalization of the ZEOS token which in turn depends not only on the *price* of ZEOS but also on the *amount* of ZEOS tokens in circulation. Burning ZEOS tokens first and foremost means burning ZEOS valuation. It takes a while for scarcity to propagate through the markets and to be reflected in a higher token price. Those dynamics are important to consider when it comes to protocol governance. Possible initial threshold values could be $L_{min} = 100$ and $L_{max} = 10$.

THE TOKEN

The entire ZEOS ecosystem is powered by the ZEOS token which fulfills multiple roles. On the one hand, it serves as a fee token for private transactions without EOS account. Since all applications on EOS could potentially integrate ZEOS private deposits and withdrawals, the ZEOS Proxy could potentially provide anonymous access to all applications on EOS. ZEOS becomes a ‘gas token’ in which fees for private transactions have to be paid for to cover the costs of EOS blockchain resources like CPU, NET and RAM. This includes private token transfers between users as well as private deposits and withdrawals to or from third-party applications on EOS. This gives ZEOS a similar character to ‘Ether’ which is the gas token of the Ethereum blockchain.

Furthermore ZEOS is the so-called 'share token' required to (fractional-)algorithmically mint zAssets. It's elastic token supply allows for capital efficient synthetic assets to be created adding a new asset class to the DeFi ecosystem on EOS. Demand for zAssets means demand for the share token ZEOS – which gets burned in order to create them.

More DeFi protocols might be added in the future potentially adding even more utility to the ZEOS token.

Finally ZEOS is the governance token of the smart organization in charge of the ecosystem. The governance tokenomics have not been finalized yet. Details will follow in a future version of this paper. It will most likely include a staking requirement for members of the organization.

Attractive staking rewards are planned to drive value back to ZEOS token holders who shoulder the entire risk of zAssets. Possible income streams are:

- ***fees from private transactions without EOS account***
- ***fees from minting/burning zAssets***
- ***fees from a deep liquidity ZEOS DEX***
- ***fees from other potential DeFi protocols build on ZEOS (eg. lending)***

While 90% of the income could be distributed to ZEOS stake holders 10% could be kept by the smart organization to fund development. However, staking tokenomics have not been finalized yet.

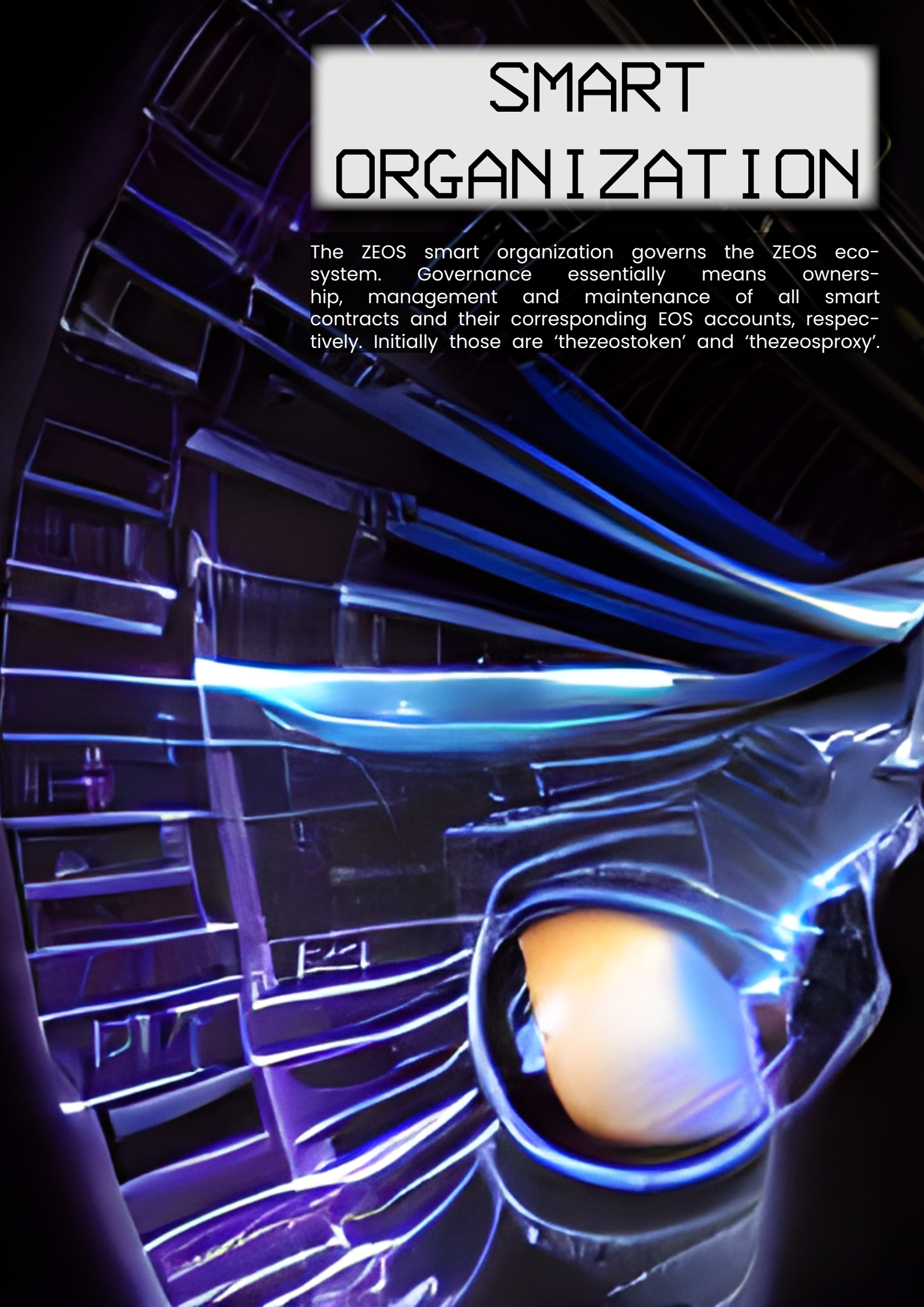
The total supply of ZEOS tokens in circulation – as well as the total supply of all zAssets in circulation – is fully transparent and auditable at any time. The balance of all ZEOS tokens which have been minted or burned in exchange for zAssets is tracked by the contract and is therefore auditable as well.

AIRDROP

The initial token supply of ZEOS amounts to almost one billion. While 90% of the tokens have been airdropped to PEOS token holders, 10% of the supply have been kept as incentive for the developer.

All EOS accounts which held at least 0.1 PEOS on December 24th 2021 received 4.76 times the amount of ZEOS tokens via airdrop on December 25th 2021. The following accounts were included in the PEOS token snapshot but were excluded from the ZEOS airdrop because they had been identified by the community to be either exchange accounts, block.one or PEOS project accounts. The surplus of ZEOS tokens resulting from the exclusion of those accounts was redistributed to PEOS token holders which is what led to the high ratio of almost 5 ZEOS dropped per 1 PEOS held at snapshot.

- hitbtcpayout
- eoshoowallet
- binancecleos
- hitbtcpayins
- newdexpublic
- hotbitioeos2
- otcbtcdotcom
- bitfinexdep1
- zbeoscharge1
- krakenkraken
- qpalzmwoskxg
- heztanrqgene
- eospstotoken
- peosmarketin
- okbtothemoon
- peosteamfund
- bitfinexcw55
- thepeostoken
- okexoffiline
- fepxecwzm41t
- binancecold1
- wlqdprkffody
- b1



SMART ORGANIZATION

The ZEOS smart organization governs the ZEOS ecosystem. Governance essentially means ownership, management and maintenance of all smart contracts and their corresponding EOS accounts, respectively. Initially those are ‘thezeostoken’ and ‘thezeosproxy’.

This implies total control by the organization over all protocol settings, resources and assets locked.

The smart organization is responsible for maintaining the ZEOS smart contracts. It is up to the organization to adopt the open source code updates from the official ZEOS GitHub repositories. The job of the ZEOS smart organization is to configure the ZEOS smart contracts and protocols responsibly in order to maximize the value of the ZEOS token. Tasks include but are not limited to:

- setting the premiums that are paid in addition to the gas fees for miner incentive and ZEOS fund
- setting L_{min} and L_{max} responsibly
- adding/removing new types of zAssets to the protocol as well as the corresponding price oracles

As owner of the ZEOS token contract the organization is also responsible for the DAPP resources and packages booked by the protocol. Particularly, governance is required to add or remove DSP packages. Ideally, more DSPs are being added over time to further decentralize VRAM, VCPU and oracle services as the DAPP network grows.

More details about the structure and exact setup of the smart organization will be provided in a future version of this paper. Most likely, members will be required to stake a certain amount of ZEOS to the organization in order to be able to participate in governance. The Vigor DAC might serve as a role model for the ZEOS smart organization. Fractal Democracy might be adopted as a consensus mechanism for the members of the organization.