otichain

#yellowpaper

www.otichain.com

OTIChain Blockchain Yellowpaper

Abstract

Blockchain-enabled smart contracts that employ proof-of-stake validation for transactions, promise significant performance advantages compared to proof-of-work solutions. For broad industry adoption, other important requirements must be met in addition. For example, stable backwards-compatible smart-contract systems must automate cross-organizational information-logistics orchestration with lite mobile wallets that support simple payment verification (SPV) techniques. The currently leading smart-contract solution Ethereum, uses computationally expensive proof-of-work validation, is expected to hard-fork multiple times in the future and requires downloading the entire blockchain. Consequently, Ethereum smart contracts have limited utility and lack formal semantics, which is a security issue.

1 Introduction

Orchestration and choreography protocols that facilitate, verify and enact with computing means a negotiated agreement between consenting parties, are termed smart contracts. The latter initially find application in diverse domains such as, e.g., financial-technology, Internet-of-Things (IoT) applications, digital-signing solutions. An essential aspect of smart contracts is a decentralized validation of transactions, initially by means of so-called proof-of-work (PoW). The core technology that enables smart contracts is a public distributed ledger termed the blockchain, which records transaction events without requiring a trusted central authority. Blockchain technology spreads in popularity with the inception of Bitcoin, a peer-to-peer (P2P) cryptocurrency and payment system that comprises a limited set of operations on the protocol layer. Bitcoins use PoW for transaction validation that is computationally expensive and electricity intensive. In contrast to Bitcoins, many smart-contract systems are equipped with the Turing-complete language Solidity that resembles JavaScript syntax and targets for enactment, like the Ethereum Virtual machine. Ethereum is the de-facto leading smart-contract system despite being plagued by several deficiencies. First, proof-of-work transaction validation diminishes scalability to the point where Ethereum is considered to not be feasible for most industry applications. Second, in a recent crowdfunding case study, the Ethereum affiliated Solidity smart contract was hacked because of security flaws resulting from a lack in the state of the art with respect to tools for formal verifications. The security flaw resulted in a loss of ca. $50 million. Consequently, Ethereum performed a hardfork resulting in a schism yielding two separate Ethereum versions. Yet another Ethereum hardfork was caused by a denial of service attack, and more hardforks must be expected for realizing proof-of-stake transaction validation and blockchain sharding. More reasons limit widespread Ethereum industry adoption. For example, an inability to automate cross-organizational information-logistics, lacking privacy protecting differentiations between external versus related internal private contracts, secure and stable virtual machines for blockchains with better performing proof-of-stake transaction validation, formally verifiable smart-contract languages, lite wallets that do not require downloading the entire blockchain, and mobile-device solutions for smart contracts with simple payment verification (SPV). The latter means that clients merely download block headers when they connect to an arbitrary full node. While OTIChain uses the Ethereum Virtual Machine (EVM) for a current lack of more suitable alternatives, the EVM has deficiencies such as earlier experienced attacks against mishandled exceptions and against dependencies such as for transaction-ordering, timestamps, and so on. It is also desirable for a smart-contract system to achieve industry-scalability with employing sidechains and unspent transaction outputs (UTXO), achieving compatibility to other blockchain systems such as Bitcoin. Furthermore, an adoption of features from the Bitcoin Lightning Network yields scalability via bidirectional micropayment channels. While smart-contract systems such as Ethereum attract attention, a widespread industry adoption does not exist for the above discussed reasons.

2 OTIChain Performance Advantage

OTIChain is a UTXO-based smart-contract system with a proof-of-stake (PoS) consensus model. The latter means the creator of the next block is chosen based on the held wealth in cryptocurrency. Thus, blocks are usually forged, or minted instead of being mined, there are block rewards instead of transaction fees and forgers receive a percentage of "interest" for the amount of funds they stake. OTIChain is compatible with the Bitcoin and Ethereum ecosystems and aims at producing a variation of Bitcoin with Ethereum Virtual Machine (EVM) compatibility. Note that differently to Ethereum, the OTIChain EVM is constantly backwards compatible. Pursuing a pragmatic design approach, OTIChain employs industry use cases with a strategy comprising mobile devices. The latter allows OTIChain promoting blockchain technology to a wide array of Internet users and thereby, decentralizing PoS transaction validation.

2.1 Bitcoin UTXO Versus Ethereum Account Model

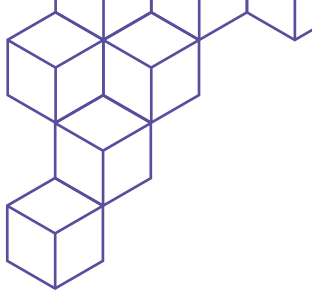
In the UTXO model, transactions use as input unspent Bitcoins that are destroyed and as transaction outputs, new UTXOs are created. Unspent transaction outputs are created as change and returned to the spender. In this way, a certain volume of Bitcoins are transferred among different private key owners, and new UTXOs are spent and created in the transaction chain. The UTXO of a Bitcoin transaction is unlocked by the private key that is used to sign a modified version of a transaction. In the Bitcoin network, miners generate Bitcoins with a process called a coinbase transaction, which does not contain any inputs. Bitcoin uses a scripting language for transactions with a limited set of operations. In the Bitcoin network, the scripting system processes data by stacks (Main Stack and Alt Stack), which is an abstract data type following the LIFO principle of Last-In, First-Out. In the Bitcoin client, the developers use **isStandard()** function to summarize the scripting types. Bitcoin clients support: **P2PKH** (Pay to Public Key Hash), **P2PK** (Pay to Public Key), **MultiSignature** (less than 15 private key signatures), **P2SH** (Pay to Script Hash), and **OP_RETURN**. With these five standard scripting types, Bitcoin clients can process complex payment logics. Besides that, a non-standard script can be created and executed if miners agree to encapsulate such a non-standard transaction. For example, using **P2PKH** for the process of script creation and execution, we assume paying 0.01BTC for a coffee in a Bar with the imaginary Bitcoin address "Coffee Address". The output of this transaction is:

OP_DUP OP_HASH160 <Coffee Public Key Hash> OP_EQUAL OP_CHECKSIG

The operation **OP_DUP** duplicates the top item in the stack.
OP_HASH160 returns a Bitcoin address as top item. To establishes ownership of a bitcoin, a Bitcoin address is required in addition with a digital key and a digital signature.
OP_EQUAL yields **TRUE (1)** if the top two items are exactly equal and otherwise **FALSE (0).**
Finally, **OP_CHECKSIG** produces a public key and signature together with a validation for the signature pertaining to hashed data of a transaction, returning **TRUE** if a match occurs.
The unlock script according to the lock script is:

<Coffee Signature> <Coffee Public Key>

The combined script with the above two:



<Coffee Signature> <Coffee Public Key> OP_DUP OP_HASH160
<Coffee Public Key Hash> OP_EQUAL OP_CHECKSIG

Only when the unlock script and the lock script have a matching predefined condition, is the execution of the script combination true. It means, the Coffee Signature must be signed by matching the private key of a valid Coffee Address signature and then the result is true. Unfortunately, the scripting language of Bitcoin is not Turing-complete, so there is no loop function. The Bitcoin scripting language is not a commonly used programming language. The limitations mitigate the security risks by preventing the occurrence of complex payment conditions, like generating infinite loops or other complicated logic loopholes. In the UTXO model, it is possible to transparently trace back the history of each transaction through the public ledger. The UTXO model has parallel processing capability to initialize transactions among multiple addresses indicating the extensibility. Additionally, the UTXO model supports privacy in that users can use Change Address as the output of a UTXO. The target of OTIChain is to implement smart contracts based on the innovative design of the UTXO model, instead of an account based system, like Ethereum. More precisely, each account experiences direct value and information transfers with state transitions. An Ethereum account address of 20 bytes comprises a nounce as a counter for assuring one-time processing for a transaction, the balance of the main internal crypto fuel for paying transaction fees called Ether, an optional contract code and default-empty account storage. The two types of Ether accounts are on the one hand, private-key controlled external and on the other hand, contract-code controlled. The former code-void account type creates and signs transactions for message transfer. The latter activates code after receiving a message for reading and writing internal storage, creating contracts, or sending other messages. In Ethereum, balance management resembles a bank account in the real world. Every newly generated block potentially influences the global status of other accounts. Every account has its own balance, storage and code-space base for calling other accounts or addresses, and stores respective execution results. In the existing Ethereum account system, users perform P2P transactions via client remote procedure calls. Although sending messages to more accounts via smart contracts is possible, these internal transactions are only visible in the balance of each account and tracking them on the public ledger of Ethereum is a challenge. We consider the Ethereum account model to be a scalability bottleneck and see clear advantages of the Bitcoin-network UTXO model. Since the latter enhances the network effect we wish to offer, the design decision for the OTIChain blockchain is the adoption of the UTXO model.

2.2 Consensus Management

Cryptocurrencies have managed to change the way finance and money is defined. The advent of Bitcoin has showed how a peer-to-peer network can prevent forgery by solving the "Byzantine Generals Problem." Since then many different coins have been created based on Bitcoin's open source code. There are two major methods to secure the blockchain network. The first is "Proof of Work" and the second is "Proof of Stake". The theory behind PoW is to hold a mathematical competition. The first computer to solve the puzzle is the one that confirms the block of transactions and wins the coin reward of it. This is called mining. However, this creates a problems of wasted high resource cost, wasted high energy cost, high fees, slowness (slow processing of transactions, slow tx/sec) and ends up centralizing the network into a few pools of computers owned by a few rich persons that could afford all the hardware bills, all because of the very nature of mining. In the Bitcoin network, miners participate in the verification process by hash collision through PoW. When

the hash value of a miner is able to calculate and meet a certain condition, the miner may claim to the network that a new block is mined:

$$\text{Hash(BlockHeader)} \leq \frac{M}{D}$$

For the amount of miners M and the mining difficulty D, the **Hash()** represents the SHA256 power with value range [0, M], and D. The SHA256 algorithm used by Bitcoin enables every node to verify each block quickly, if the number of miners is high versus the mining difficulty. The 80 byte BlockHeader varies with each different Nonce. The overall difficulty level of mining adjusts dynamically according to the total hash power of the blockchain network. When two or more miners solve a block at the same time, a small fork happens in the network. This is the point where the blockchain needs to make a decision as to which block it should accept, or reject. In the Bitcoin network, the chain is legitimate that has the most proven work attached. On the other hand, PoS is a competition between coinholders, where based on connectivity to the network and random chance, you can confirm a block of transactions and receive coin rewards. This is called staking. It doesn't necessitate any particular hardware except a normal computer with an internet connect and you are guaranteed to be rewarded proportionally the amount of coins you hold which makes it fair and decentralized. Coin rewards are determined by a yearly supply inflation and awarded proportionally to addresses that stake (equivalent of mining in PoS). Most PoS blockchains can source their heritage back to PeerCoin that is based on an earlier version of Bitcoin Core. There are different PoW algorithms such as Scrypt, X11, Groestl, Equihash, etc. The purpose of launching a new algorithm is to prevent the accumulation of computing power by one entity and ensure that Application Specific Integrated Circuits (ASIC) can not be introduced into the economy. In a traditional PoS transaction, the generation of a new block must meet the following condition:

$$\text{ProofHash} < \text{coins} * \text{age} * \text{target}$$

In ProofHash, the stake modifier computes together with unspent outputs and the current time. With this method, one malicious attacker can start a double-spending attack by accumulating large amounts of Coin Age. Another problem caused by Coin Age is that nodes are online intermittently after rewarding instead of being continuously online. The original PoS implementation suffers from several security issues due to possible Coin Age attacks, and other types of attacks. Indeed the whole purpose of holding competitions for coins is to avoid attacks. Confirmation of transactions is an honor given to the winner of a block. Although if this system can be gamed, then it is flawed. In PoS, you first prove you have access to coins and from that point you can compete to win blocks randomly. The more people competing the more secure the block. Coin Age is the idea that the longer you hold coins the higher the probability you can win a block. Its original intention was to incentive dormant holders of coins. However, this does not encourage a node to stay connected to the network in practice since they can wait for the reward to increase. Also, coinholders can disconnect from the network for long periods of time, then reconnect and win enough blocks to risk a 50% attack on the network. The time calculation will effect payouts discouraging connectivity. Also, the fewer the nodes that are connected, the easier it is to gain a majority of the blocks forging consensus. Also, stakes can be computed in advance to make the attack more effective. Timestamps are used in PoS to get a general idea of time. Drift calculations are used to prevent forging erroneous timestamps. In PoW, a difficulty increase or decrease is made depending on how quickly a block was

produced. However, as a precautionary method to prevent any sort of "Timing Attacks" PoS development teams use centralized checkpoints. Therefore, in the improved version of PoS agreement, Coin Age removal encourages more nodes to be online simultaneously. OTIChain is built upon an evolution of “Proof-of-Stake Version 3”, an improvement over version 2 that was also implemented in the Blackcoin project. This type of Proof of Stake (PoSv3) is built for UTXO based blockchains. Instead of a coinbase transaction in a PoW system, the OTIChain PoS system has a coinstake transaction (2nd transaction in the block, 1st being an empty coinbase). The coinstake transaction has some specific rules it must abide by, and each block must have exactly one staking transaction. Secondly, the block timestamp must have the bottom four bits set to 0 so the blocktime can only be represented in 15-second intervals. Each UTXO can only be used once every ~48 hours to produce a staking transaction.

The Kernel Hash is built from the following data:

- **Previous block’s “stake modifier”** (hash of prevout transaction in PoS blocks and previous block’s stake modifier)
- **Timestamp from “prevout” transaction** (the transaction output that is spent by the first vin of the staking transaction)
- The **hash of the prevout transaction**
- The **output number of the prevout** (ie, which output of the transaction is spent by the staking transaction)
- **Current block time**, with the bottom 4 bits set to 0 to reduce granularity. This is the only thing that changes during staking process

Secondly, PoS blocks timestamps must have certain granularity within 15-second target blocktime. Also, each UTXO is eligible to produce one rewarded block in 48h, the rest of blocks staked from the same UTXO are not rewarded. The PoS difficulty ends up being twice as easy to achieve when staking 2 coins, compared to staking just 1 coin. If this were not the case, then it would encourage creating many tiny UTXOs for staking, which would bloat the size of the blockchain and ultimately cause the entire network to require more resources to maintain, as well as potentially compromise the blockchain’s overall security. OTIChain proof-of-stake algorithm is uniquely suited to its UTXO blockchain base. For example, the Ethereum blockchain is looking into Casper, a PoS protocol implemented as a set of smart contracts on top of the existing Ethereum blockchain. There is a system of punishments and likelihood of loosing your stake if you submit bad blocks and a minimum requirement as a deposit.

2.3 Nothing-At-Stake problem, OTIChain versus Ethereum

One of the problems that Casper tries to work on is the “nothing-at-stake” problem, described as this in the Ethereum wiki: *However, this algorithm has one important flaw: there is “nothing at stake”. In the event of a fork, whether the fork is accidental or a malicious attempt to rewrite history and reverse a transaction, the optimal strategy for any miner is to mine on every chain, so that the miner gets their reward no matter which fork wins. Thus, assuming a large number of economically interested miners, an attacker may be able to send a transaction in exchange for some digital good (usually another cryptocurrency), receive the good, then start a fork of the blockchain from one block behind the transaction and send the money to themselves instead, and even with 1% of the total stake the attacker’s fork would win because everyone else is mining on both.*

Casper deals with the nothing-at-stake problem using a punishment mechanic and a complex series of crypto economics incentive schemes. For OTIChain, the nothing-at-stake problem will be mitigated using a whole new Reward Governance Protocol. Since stakers earns a % of stake interest per coin/every two days, the blockchain itself is being able to generate new blocks with no reward to keep OTIChain blockchain moving. Every new block is generated, validated and included inside our blockchain every 15 seconds, allowing OTIChain to speed up the transactions confirmation times and the Smart Contract System. PoS solves PoW problems of Bitcoin as it manages to be fast and low cost while remaining decentralized.

2.4 Blockchain Precomputation

The block timestamp is key to the PoS system. It is possible in theory to fork a coin by changing previous timestamps. The stake modifier does not obfuscate the hash of sufficiently to prevent knowing future proofs. So an attacker can attempt to compute all of the blocks in advance and run a higher probability to forge multiple consecutive blocks. In OTIChain blockchain, the stake modifier is changed at every modifier interval to better obfuscate any calculations that would be made to pinpoint the time for the next proof-of-stake.

2.5 Multisignature & Cold Staking

The final noteworthy addition to the protocol was the implementation of "Multisignature Staking". One drawback to many staking algorithms is they only support staking with a single key. Since the popularity and use of Bitbay which uses a two party escrow system also known as "Double Deposit Escrow" and extremely secure dual key accounts, it has become important to allow these accounts to participate in securing the network. Beyond dual key accounts there are many other types of inputs that make use of p2sh and lock times and those must also be allowed to secure the network as well. The other problem is that in a single key account, a hacker can use key-loggers to obtain your password and compromise your wallet while it is unlocked for staking. OTIChain allows users to place the block signing key in the output of "6a" known as a burn address so they can stake by sending a standard transaction. This allows any input to be eligible for submission. This gives OTIChain a huge advantage for custom staking software, voting and the legendary "Cold Staking". The "Cold Staking" technique involves multiple computers. Basically when a multisignature input is eligible for staking, the signatures are split up between many computers. This makes an account virtually impossible to hack because even if a single key was compromised, the other keys are in a completely different location either on the local area network or on multiple servers.

2.6 UTXO Stake Eligibility

The Block Reward in most PoS systems is unfortunately based on Coin Age. In theory, this is to distribute interest fairly by allowing nodes to receive latent payments due. However, this system does not work because nodes can stay disconnected and with many split inputs, reconnect to the network and game the reward system. Also, it does not give nodes any incentive to stay connected. In a decentralized system, the more nodes connected the better the security since it shifts trust from a single entity to the network itself. OTIChain introduces a unique block reward. A mature UTXO (depending on the current blockchain state, difficulty and blocktime) is eligible for generating a staking transactions every ~48 hours. The staked amount gives a reward approx. of *0,00013698 OTI * STAKED UTXO per day.* To prevent tiny inputs (dust) to stake and slow down the

daemons and wallets, a UTXO is eligible for staking only if consistent. Meaning that inputs with a relative low amount of OTI will not be able to generate any staking transaction.

2.5 UTXO Splitting and Dust preventing System

Since UTXO Model of OTI cryptocurrency was directly based on OTICHAIN blockchain, we have developed a way to allow enough inputs to be "at stake" by directly splitting the staked transactions. OTI integrates a whole new and advanced system to prevent dust slowing down user wallets. Defined by Consensus rules and on demand (for example by issuing a command or click a button), every matured UTXOs with value under given limit will be combined and will produce a single output, getting a final value which is the sum of those inputs. PoS 3.1 rewards investors that stake their coins longer, while giving no incentive to coin holders who leave their wallets offline.

3 OTIChain Contract and EVM Integration

The EVM is stack-based with a 256-bit machine word. Smart contracts that run on Ethereum use this virtual machine for their execution. The EVM is designed for the blockchain of Ethereum and thus, assumes that all value transfer use an account-based method. OTIChain is based on the blockchain design of Bitcoin and uses the UTXO-based model. Thus, OTIChain has an account abstraction layer that translates the UTXO-based model to an account-based interface for the EVM. Note that an abstraction layer in computing is instrumental for hiding the implementation details of particular functionality to establish a separation of concerns for facilitating interoperability and platform independence.

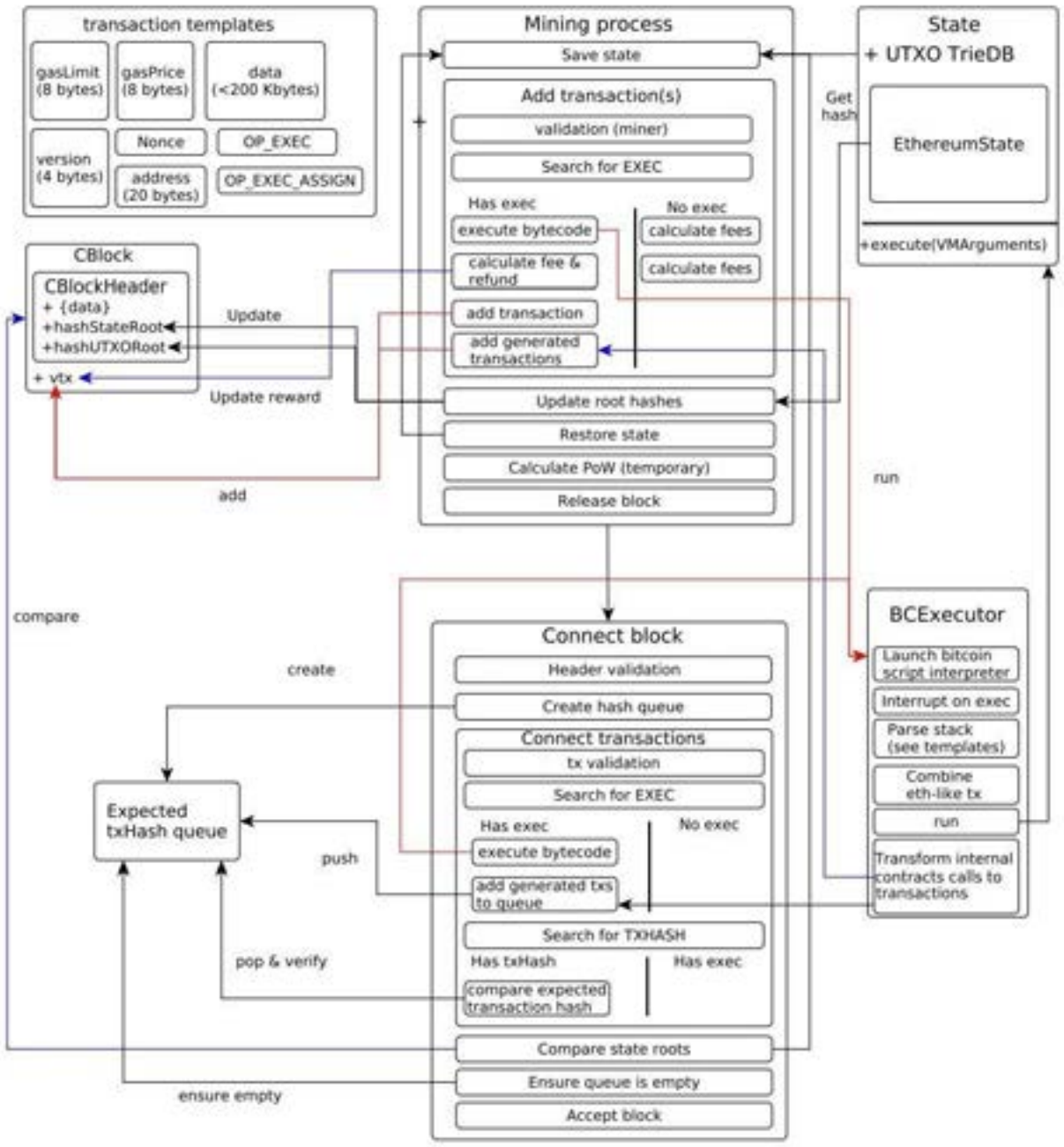
3.1 EVM Integration

All transactions in OTIChain use the Bitcoin Scripting Language, just like Bitcoin. In OTIChain however, there exist three new opcodes:

- **OP_EXEC**: This opcode triggers special processing of a transaction and executes specific input EVM bytecode.
- **OP_EXEC_ASSIGN**: This opcode also triggers special processing such as **OP_EXEC**.
  This opcode has as input a contract address and data for the contract. Next follows the execution of contract bytecode while passing in the given data (given as **CALLERDATA** in EVM). This opcode optionally transfers money to a smart contract.
- **OP_TXHASH**: This opcode is used to reconcile an odd part of the accounting abstraction layer and pushes the transaction ID hash of a currently executed transaction.

Traditionally, scripts are only executed when attempting to spend an output. For example, while the script is on the blockchain, with a standard public key hash transaction, no validation or execution takes place. Execution and validation does not happen until a transaction input references the output. At this point, the transaction is only valid if the input script (**ScriptSig**) does provide valid data to the output script that causes the latter to return non-zero. OTIChain however, must accommodate smart contracts that execute immediately when merged into the blockchain. As depicted in Figure 1 , OTIChain achieves this by the special processing of transaction output scripts (**ScriptPubKey**) that contain either **OP_EXEC**, or **OP_EXEC_ASSIGN**. When one of these opcodes is detected in a script, it is executed by all nodes of the network after the transaction is placed into a block. In this mode, the actual Bitcoin Script Language serves less as a scripting language and instead carries data to the EVM. The latter changes state within its own state database, upon execution by

either of the opcodes, similar to a Ethereum contract. For easy use of OTIChain smart contracts, we have to authenticate the data sent to a smart contract as well as its creator stemming from a particular pub-keyhash address. In order to prevent the UTXO set of the OTIChain blockchain from becoming too large, **OP_EXEC** and **OP_EXEC_ASSIGN** transaction outputs are also spendable. **OP_EXEC_ASSIGN** outputs are spent by contracts when their code sends money to another contract, or to a pubkeyhash address. **OP_EXEC** outputs are spent whenever the contract uses the suicide operation to remove itself from the blockchain.



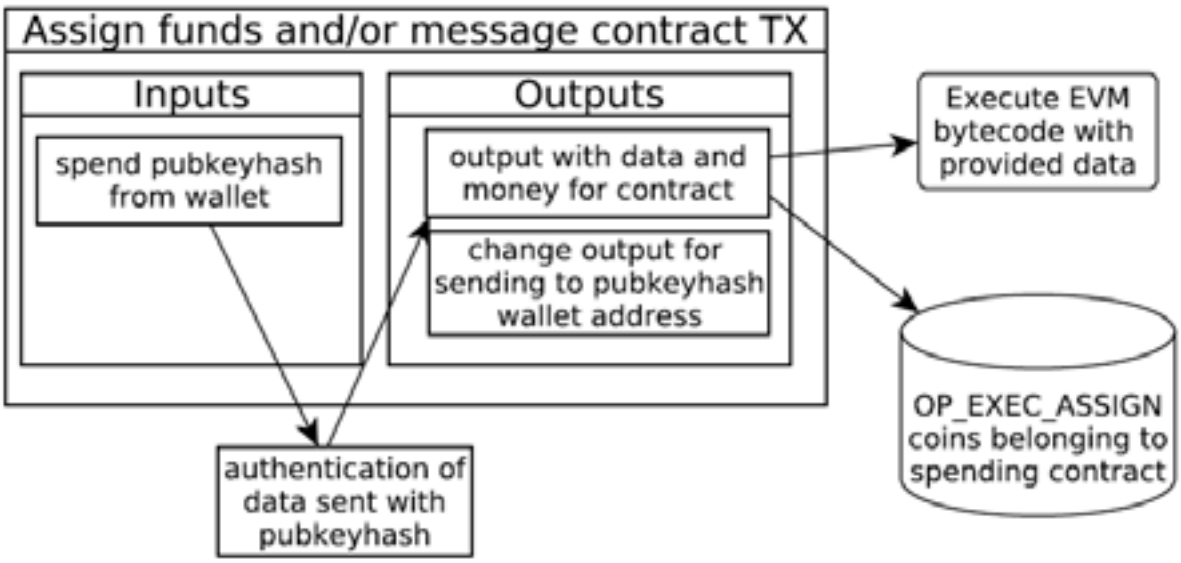
3.2 OTIChain Account Abstraction Layer

The EVM is designed to function on an account-based blockchain. OTIChain however, being based on bitcoin, uses a UTXO-based blockchain and contains an Account Abstraction Layer (AAL) that allows the EVM to function on the OTIChain blockchain without significant modifications to the

virtual machine and existing Ethereum contracts. The EVM account model is simple to use for smart-contract programmers. Operations exist that check the balance of the current contract and other contracts on the blockchain, and there are operations for sending money (attached to data) to other contracts. Although these actions seem fairly basic and minimalistic, they are not trivial to apply within the UTXO-based OTIChain blockchain. Thus, the AAL implementation of these operations may be more complex than expected. A OTIChain-blockchain deployed smart contract is assigned and callable by its address and comprises a newly deployed contract balance set to zero. There is currently no protocol in OTIChain that allows a contract to be deployed with a non-zero balance. In order to send funds to a contract, a transaction uses the **OP_EXEC_ASSIGN** opcode.
The example output script below sends money to a contract:

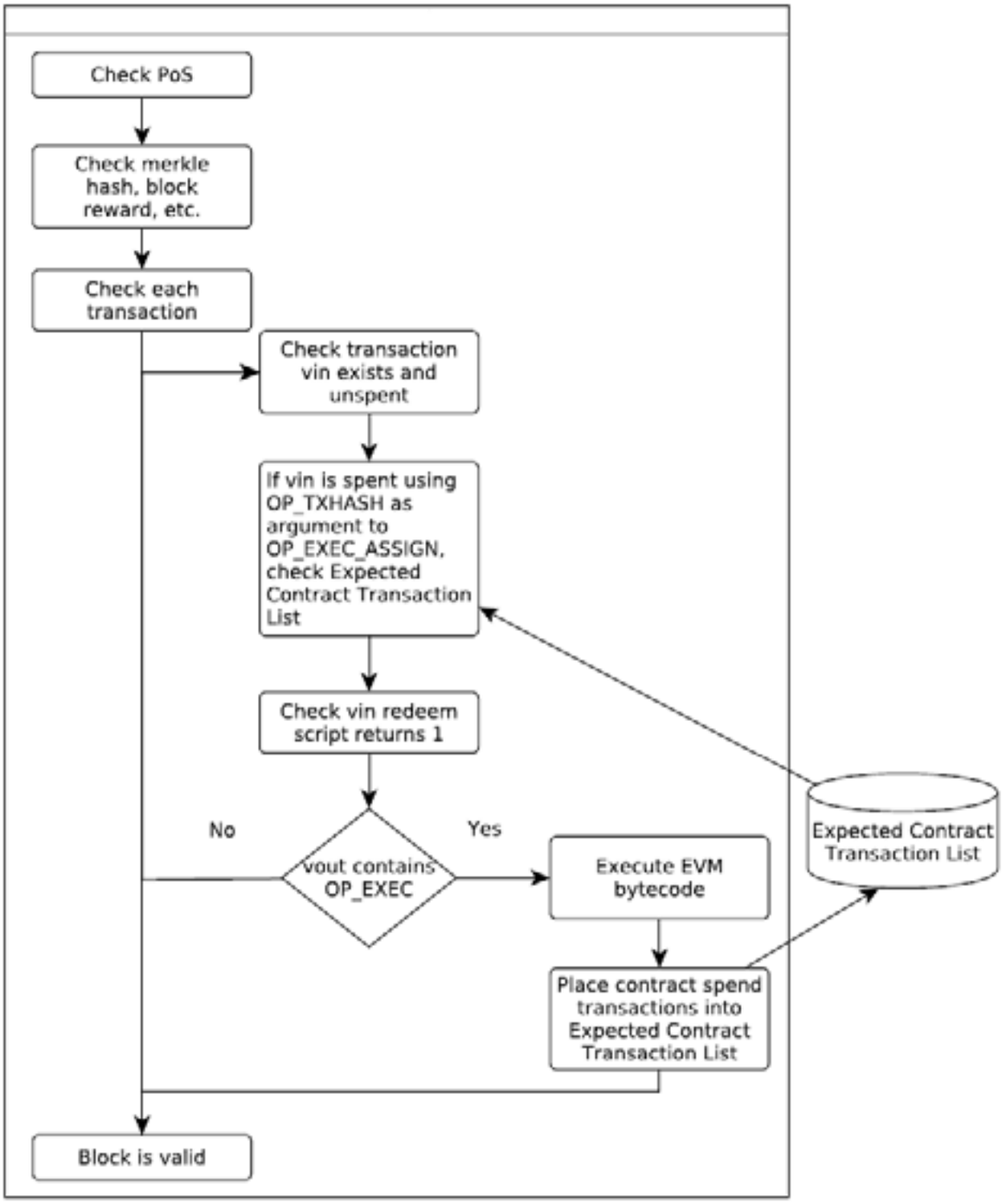
1 ; the version of the VM
10000; gas limit for the transaction
0,0001; gas price in OTI
0xE1A5; data to send the contract (usually using the Solidity ABI)
0x30f1ab168a01c48ccac04d164dc8601c5802927f; ripemd-160 hash of contract txid
OP_EXEC_ASSIGN

The simple script above hands over transaction processing to the **OP_EXEC_ASSIGN** opcode. Assuming no out-of-gas, or other exceptions occur, the value amount given to the contract is **OutputValue**. By adding this output to the blockchain, the output enters the domain of the contract owned UTXO set. This output value is reflected in the balance of the contract as the sum of spendable outputs.



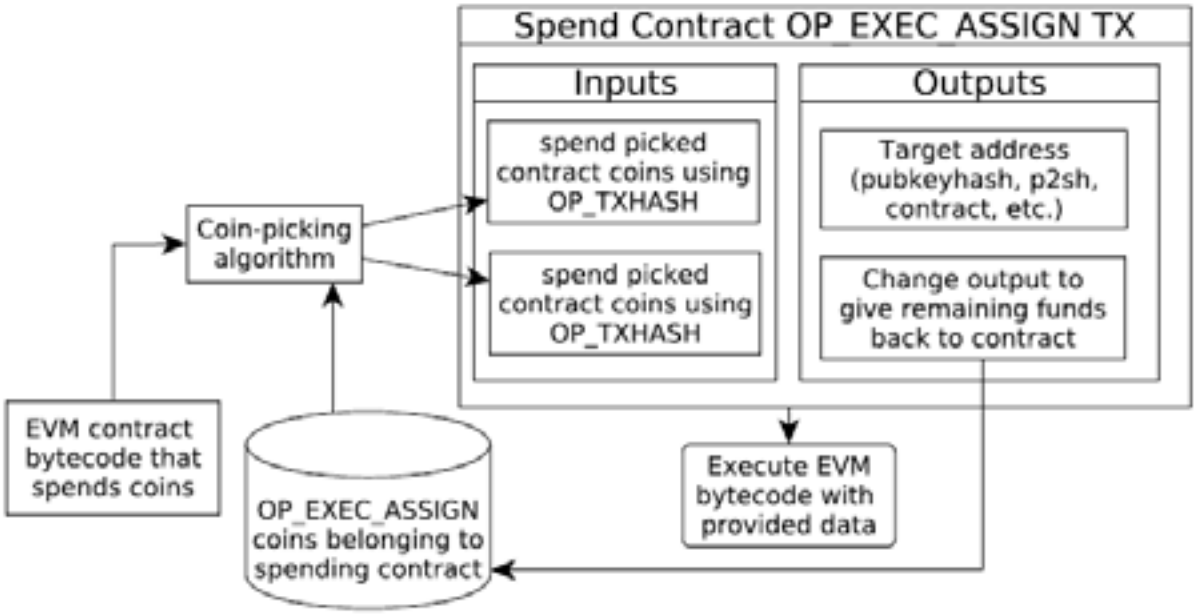
When the contract sends funds to another contract or public key hash address, the former spends one of its owned outputs. The sending contract involves Expected Contract Transactions for the fund sending. These transactions are special in that they must exist in a block to be valid for the OTIChain network. Expected Contract Transactions are generated by miners while verifying and executing

transactions, rather than being generated by consumers. As such, they are not broadcast on the P2P network.



The primary mechanism to perform Expected Contract Transactions is the new opcode, **OP_TXHASH**. Internally, both **OP_EXEC** and **OP_EXEC_ASSIGN** have two different modes. Upon their execution as part of the output script processing, the EVM is executed. When the opcodes are executed as part of input script processing, however, the EVM is not executed to avoid double execution. Instead, the **OP_EXEC** and **OP_EXEC_ASSIGN** opcodes behave similar to no-ops and

return either 1 or 0, i.e., spendable or not spendable respectively, based on a given transaction hash. This is why **OP_TXHASH** is so important to the functioning of this concept. Briefly, **OP_TXHASH** is a new opcode added which pushes the current spending transaction's SHA256 hash onto the Bitcoin Script stack. The **OP_EXEC** and **OP_EXEC_ASSIGN** opcodes check the Expected Contract Transaction List during a spend attempt. After the transaction passes (usually from **OP_TXHASH**) to the opcodes that exist in the Expected Contract Transaction List, the result is 1, or spendable. Otherwise, the return is 0, or not spendable. In this way, **OP_EXEC** and **OP_EXEC_ASSIGN** using vouts are only spendable when a contract and thus, the Account Abstraction Layer, requires that the vout is spendable, i.e., while the contract attempts sending money. This results in a secure and sound way of allowing contract funds to be spent only by a respective contract in alignment with a normal UTXO transaction. A specific scenario occurs if a contract has more than one output that can be spent. Each node may pick different outputs and thus, use completely different transactions for spending **OP_EXEC_ASSIGN** transactions. This is resolved in OTIChain by a consensus-critical coin picking algorithm. The latter is similar to the standard coin picking algorithm used within a user wallet. However, OTIChain significantly simplifies the algorithm to avoid the risk of denial of service (DoS) attack vectors and to realize simple consensus rules. With this consensus-critical coin picking algorithm, there is now no possibility for other nodes to pick different coins to be spent by a contract. Any miner/node who picks different outputs must fork away from the main OTIChain network, and their blocks are rendered invalid. When an EVM contract sends money either to a **pubkeyhash** address, or to another contract, this event constructs a new transaction.



The consensus-critical coin-picking algorithm chooses the best owned outputs of the contract pool. These outputs are spent as inputs with the input script (**ScriptSig**) comprising a single **OP_TXHASH** opcode. The outputs are thus, the destination for the funds, and a change output (if required) to send the remaining funds of the transaction back to the contract. This transaction hash is added to the Expected Contract Transaction List and then the transaction itself is added to the block immediately after the contract execution transaction. Once this constructed transaction is validated

and executed, a confirmation check of the Expected Contract Transaction List follows. Next, this transaction hash is removed from the Expected Contract Transaction List. Using this model, it is impossible to spoof transactions for spending them by providing a hardcoded hash as input script, instead of using **OP_TXHASH**. The above described abstraction layer renders the EVM contracts oblivious to coin picking and specific outputs. Instead, the EVM contracts know only that they and other contracts have a balance so that money can be sent to these contracts as well as outside of the contract system to **pubkeyhash** addresses. Consequently, contract compatibility between OTIChain and Ethereum is strong and very few modifications are required to port an Ethereum contract to the OTIChain blockchain.

3.3 Added Standard Transaction Types

The following are the standard transaction types that we add to OTIChain. Deploying a new contract to the blockchain requires an output script as follows:

1 ; the version of the VM
[Gas limit]
[Gas price]
[Contract EVM bytecode]
OP_EXEC

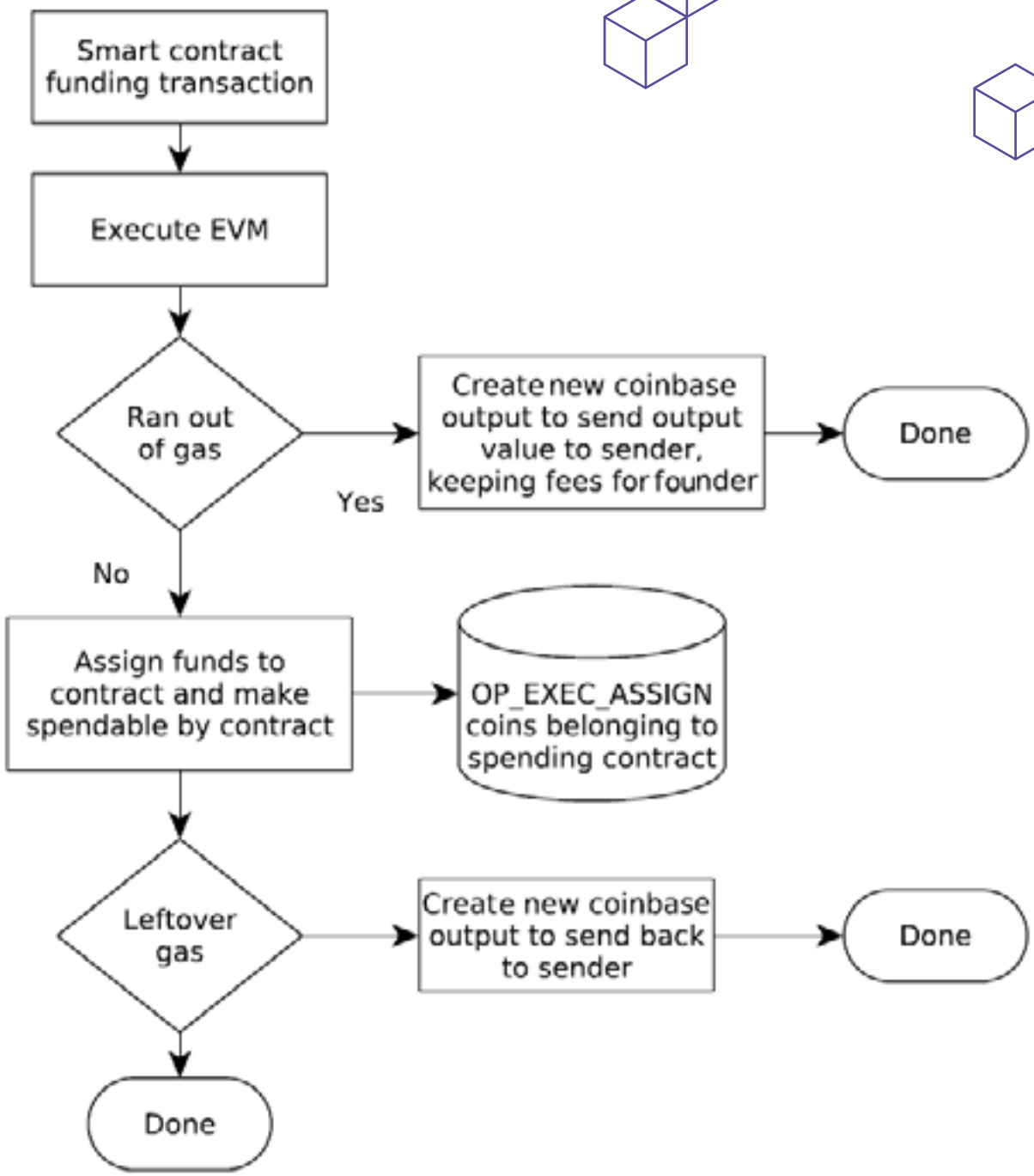
Sending funds to an already deployed contract on the blockchain requires the script below:

1 ; the version of the VM
[Gas limit]
[Gas price]
[Data to send to the contract]
[rip-emd160 hash of contract transaction id]
OP_EXEC_ASSIGN

Note there are no standard transaction types for spending as that requires the Expected Contract Transaction List. Thus, these spending transactions are neither broadcast nor valid on the P2P network.

3.4 Gas Model

A problem OTIChain faces with adding Turing-completeness to the Bitcoin blockchain is relying on only the size of a transaction, which is not reasonable for determining the fee paid to OTIChain Founder address. The reason is that a transaction may infinitely loop and halt the entire blockchain for transaction-processing miners. The OTIChain project adopts the concept of gas from Ethereum. In the gas concept, each EVM opcode executed has a price and each transaction has an amount of gas to spend. Post-transaction remaining gas is refunded to the sender.



When the gas required for contract execution exceeds the amount of gas available to a transaction, then the actions of a transaction and state changes are reverted. Thus, any modified permanent storage is reverted to its original state including any spending of contract funds so that the latter are not spent. Despite a reversion, all gas of a transaction is consumed and given to the processing miner since the computing resources have already been spent. Although OTIChain uses the gas model from Ethereum, we expect the gas schedule, i.e., the gas price of each EVM opcode, to significantly differ from Ethereum. The exact values are determined by comparing existing prices in Ethereum with the amount of processing and blockchain resources required for each opcode to OTIChain. When creating a contract funding, or deployment transaction, the user specifies two specific items for gas. The **GasLimit** determines the amount of consumable gas by a contract execution. The second item is the **GasPrice** to set the exact price of each unit of gas in OTI. The maximum OTIChain expenditure of a contract execution equates the multiplication of **GasLimit** by

GasPrice. If this maximum expenditure exceeds the transaction fee provided by the transaction then the latter is invalid and can not be processed. The remaining transaction fee after subtracting this maximum expenditure is the Transaction Size Fee and analogous to the standard Bitcoin fee model. To determine the appropriate priority of a transaction, OTIChain Founder considers two variables. First, the transaction size fee must match the total size of a transaction, i.e., usually determined by a minimum amount of coins per kilobyte formula. The second variable is the **GasPrice** of a contract execution. In combination, OTIChain Founder chooses the fastest and cheapest transactions to process and include in a block.

4 RPC Api

The demon (node, wallet), will be compatible with Bitcoin and Etereum RPC-Calls. The inclusion of these commands makes, in fact, the nodes and wallets of the blockchain integrable with any platform and software language, mobile, web and IoT. They are grouped into 9 macro categories:

- **Blockchain**
  callcontract, getaccountinfo, getbestblockhash, getblock, getblockchaininfo, getblockcount, getblockhash, getblockheader, getblockstats, getchaintips, getchaintxstats, getdifficulty, getmempoolancestors, getmempooldescendants, getmempoolentry, getmempoolinfo, getrawmempool, getstorage, gettransactionreceipt, gettxout, gettxoutproof, gettxoutsetinfo, listcontracts, preciousblock, pruneblockchain, savemempool, scantxoutset, searchlogs, verifychain, verifytxoutproof, waitforlogs
- **Control**
  getmemoryinfo, help, logging, stop, uptime
- **Generation**
  generate, generatetoaddress
- **Mining**
  getblocktemplate, getmininginfo, getnetworkhashps, getstakinginfo, getsubsidy, prioritisetransaction, submitblock
- **Network**
  addnode, clearbanned, disconnectnode, getaddednodeinfo, getconnectioncount, getnettotals, getnetworkinfo, getpeerinfolistbanned, ping, setban, setnetworkactive
- **RAW Transaction**
  combinepsbt, combinerawtransaction, converttopsbt, createpsbt, createrawtransaction, decodepsbt, decoderawtransaction, decodescript, finalizepsbt, fromhexaddress, fundrawtransaction, gethexaddress, getrawtransaction, sendrawtransaction, signrawtransaction, signrawtransactionwithkey, testmempoolaccept
- **Utility**
  createmultisig, estimatesmartfee, signmessagewithprivkey, validateaddress, verifymessage
- **Wallet**
  abandontransaction, abortrescan, addmultisigaddress, backupwallet, bumpfee, createcontract, createwallet, dumpprivkey, dumpwallet, encryptwallet, getaddressesbylabel, getaddressinfo, getbalance, getnewaddress, getrawchangeaddress, getreceivedbyaddress, gettransaction, getunconfirmedbalance, getwalletinfo, importaddress, importmulti, importprivkey, importprunedfunds, importpubkey, importwallet, keypoolrefill, listaccounts, listaddressgroupings, listlabels, listlockunspent, listreceivedbyaddress, listsinceblock, listtransactions, listunspent, listwallets, loadwallet, lockunspent, removeprunedfunds, rescanblockchain, reservebalance, sendmany,

sendmanywithdupes, sendtoaddress, sendtocontract, sethdseed, settxfee, signmessage, signrawtransactionwithwallet, unloadwallet, walletcreatefundedpsbt, walletlock, walletpassphrase, walletpassphrasechange, walletprocesspsbt

- **ZeroMQ** (notifications) getzmqnotifications

5 Conclusion

This whitepaper presents the OTIChain-framework as a UTXO smart-contract blockchain-technology solution. We show the specific OTIChain transaction-processing implementation that uses proof-of-stake validation. Furthermore, OTIChain integrates the Ethereum virtual machine (EVM) together with the Bitcoin unspent transaction output protocol. Note that the OTIChain EVM constantly remains backwards compatible. The adoption of proof-of-stake into OTIChain constitutes a considerable saving of computational effort over the not scaling Ethereum alternative that still uses proof-of-work. While Ethereum plans to also adopt proof-of-stake, it is unclear when such a new version will be released. Also the use of unspent transaction outputs is more scalable in comparison to the account management of Ethereum. In combination with simple payment verification, OTIChain already develops a smart-contract mobile-device solution. The elimination of Block Reward based on time was an obvious improvement. In contrary to the many PoS coins that do not have enough nodes, this PoS 3.1 feature is a great advantage to small coinholders. Although the lack of statistical data conserning PoS coins, it is self-evident that there is usually less than 20% of the coinholders staking. A reward of 0,00013698 OTI * STAKED UTXO per day it can only generate 1% of annual inflation. The change in granularity was useful to prevent "Stake Grinding". Even with all the hashing power of the Bitcoin network, using PoS a network attack in practice would be extremely unlikely to the point of being realistically not possible. PoS is one of the most secure and reliable system ever created. Everything is done to ensure anonymity, to keep as many nodes connected as possible, to guarantee decentralization and to mitigate all attacks. Decentralization was the original core ideology in Bitcoin, but sadly, Bitcoin's flaws prevented it to prevail in the long run. The entire purpose of a secure and fair financial system is to place control of it in the hands of the people so that it be for the people and by the people. OTIChain has solved the main issues of Bitcoin's PoW while ensuring its own future by providing an incentive to stay connected to the network in order to keep it secure and decentralized.

5.1 OTIChain vs Bitcoin vs Ethereum

	OTIChain	Bitcoin	Ethereum
Method	Proof-of-Stake 3.1	Proof-of-Work	Proof-of-Work
Algorithm	--	SHA256	Ethash
Network Security	Minting	Mining	Mining
Block speed	15 seconds	10 minutes	20 seconds
Maximum Supply	1% annual inflation	21 million BTC	Unlimited
Environment Friendly	Yes	No	No
Network Incentive	Annual rewards 5%	Mining based	Mining Based
Transactions fees	Very very low	Depend on network difficulty	High GasPrice in SmartContract

Legal Disclaimer

The information in this white paper is purely descriptive and not binding. Please note that this document includes forward-looking statements, statements of intent, discussion of plans, estimates or other information that could be considered forward-looking. While these forward- looking statements represent our judgment and expectations about what the future holds, they are not an offer or solicitation to purchase any product, good or service.

All statements are subject to risks and uncertainties that could cause the actual results of Otichain's development to differ. No information in this white paper has been reviewed or approved by any regulatory authority.

In addition, we intend to use the Otichain blockchain as our open source development platform, contributing these technologies under permissive licenses for the benefit of human society, not focusing merely on the profit of anyone affiliated with the project. Therefore,

otichain

www.otichain.com