QTUM TECHNICAL WHITEPAPER (draft version)

UTXO based POS Blockchain Contract and DAPP Platform

Qtum Foundation
foundation@qtum.org

DISCLAIMER

This draft whitepaper is a translation of multiple Chinese technical documents. This whitepaper is intended for release amongst a small audience, and due to the nature of the internet, we expect this to surface on forums and news sites, etc. Please keep in mind that this document is not the official Qtum whitepaper, but a pre-release intended to update individuals of what to expect. Some of the writing may be abstract or lack substance. This is not in any way a controlled document, designed to give interested parties an advantage. The official whitepaper will be released weeks before the Qtum Token Sale, and this version will be released upon request by any individual or organization.

Feedback Welcome: foundation@qtum.org

Qtum

UTXO based PoS Blockchain Contract and DAPP Platform

foundation@qtum.org

Abstract

Quantum Blockchain (‘**Qtum**’ [ˈkwɒntəm]) is a combination of Bitcoin Core, Proof-of-Stake, and the Ethereum Virtual Machine (EVM). The end product, Qtum Core, will allow smart contracts to execute on a Proof-of-Stake consensus model. This ecosystem was designed to provide a familiar environment for Smart Contract and Decentralized Application developers. The Qtum Foundation’s goal is to create a stable product, that does not require a steep learning curve, and market it for mass business adoption.

One major goal of Qtum is to build a *Value Transfer Protocol*, which, with decentralized applications, can be used to support and enhance various industries such as the financial sector, supply chain tracking, internet of things, and social media.

Another Qtum goal is to maintain compatibility with existing processes from Bitcoin and Ethereum, to be as secure as possible, and to be used easily by individuals, businesses, and developers. Qtum's future development will be guided by the community and consensus focused processes based on the existing process for Bitcoin Improvement Proposals (BIP).

Qtum is secured by the Proof-of-Stake 3.0 (POS 3.0) consensus protocol and is the first blockchain platform to integrate Proof-of-Stake technology with a Smart Contract platform. Smart contracts will also be executed as part of an Unspent Transaction Output (UTXO), which is part of the Bitcoin transactional model. This brings about many advantages:

1) Compatibility with existing Bitcoin workflows
2) Retains the privacy aspects inherent in the Bitcoin UTXO model
3) The UTXO model is scalable for the long term
4) Integration with existing alternative node models for SPV and mobile wallets
5) A simpler implementation which has been proven by Bitcoin to be secure for over eight years.

While Qtum can be used just like Bitcoin (except for the Proof-of-Stake modifications) to send and receive digital currency, the real power of Qtum is the Smart Contract system. The Ethereum Virtual Machine (EVM) has been integrated into Qtum and a blockchain abstraction layer has been placed between the EVM and the UTXO based blockchain. This way existing Ethereum smart contracts run with little or no modifications to the contract source code. At a later stage, support for more powerful virtual machines (such as ones based on LLVM, Lua, or Java) will be added as well.

"Go Mobile" is a core strategy of Qtum. The core developers of Qtum will work to ensure that third party developers have ample infrastructure and functionality to support mobile use cases. This includes a mobile wallet, API, and a framework for creating decentralized mobile applications.

Part One   Qtum Design Principles

1.1 Introduction

On 31st of October 2008, Satoshi Nakamoto announced the Bitcoin whitepaper titled “Bitcoin, A Peer to Peer Electronic Cash System”, and introduced the Bitcoin network. Bitcoin has allowed for many new ways to exchange money in a decentralized manner. However, Bitcoin users and developers have always been met with various limitations in advanced usage. The biggest limitations are that the Bitcoin Scripting Language is not Turing complete and it is severely restricted in what data it has access to. We now introduce Qtum, which rectifies this problem by directly integrating the existing Ethereum Virtual Machine (EVM) into Bitcoin’s UTXO model, allowing for Ethereum-based smart contracts to be executed on a Bitcoin-based Unspent Transaction Output (UTXO) blockchain.

There have been other methods used in the past to extend Bitcoin’s UTXO based blockchain so that it can support Turing-complete smart contracts. The most popular implementation of this has been colored coins. These coins use the existing Bitcoin network and mark certain outputs with *color* metadata, and tracks this metadata throughout the blockchain. The largest problem with this technique is that colored coin transactions can not be fully validated in a decentralized way without the entire blockchain, and thus SPV-based light wallets (such as most mobile wallets) would not function, or at least not be capable of operating in a secure decentralized manner.

Qtum uses a different route for implementing smart contracts in Bitcoin. Qtum has been created by forking Bitcoin’s source code and adding various custom modules to integrate the EVM. This strategy is used for many different reasons. Bitcoin, even with its limitations, has a very large community and has many companies who have already established business processes, workflows, and tools based on this platform. A core tenet of Qtum is for it to be compatible with existing tools and processes when possible. This will ease adoption by being familiar to both Ethereum and Bitcoin users. Additionally, creating Qtum by forking the existing Bitcoin source code allows for Qtum to be built with decreased development costs on top of a very well tested system.

Ethereum also supports smart contracts and would seem like an ideal system to build Qtum on top of; however, Ethereum and Bitcoin have a key difference in their design. Whereas Bitcoin tracks funds on the blockchain using Unspent Transaction Outputs, Ethereum uses an account-based approach. Each blockchain model has it’s own advantages and disadvantages, of course. The UTXO model was chosen for Qtum primarily because of compatibility with Bitcoin existing processes and the ability to use SPV technology to enable more mobile use cases; although, the core concepts of Ethereum are more simple.

1.2 Purpose of the Qtum Blockchain Design

Since the Bitcoin project launched, the community has built up many alternative currencies and blockchain projects. Some meaningful alternative cryptocurrency projects have become good blockchain test cases. They helped improve and mature the blockchain technology, i.e. NameCoin, etc. There are also many blockchain projects seeking to extend the limits of technology from different angles, such as the ColorCoin protocol, NXTCoin, Ripple and Stellar, BitShares, Dash, Maidsafe, Factom, and the Ethereum project which focuses on generic smart contracts and decentralized applications.

Numerous developers and community members have witnessed the fast development of blockchain technology. However, it is still facing many challenges from both technical and business perspectives.

Some of the major problems currently facing blockchain technology:

1. The lack of a business oriented smart contracts. The current smart contract platforms are mainly based on Proof-of-Work (PoW). However, the consensus mechanism of Proof-of-Stake (PoS) is more suited for business applications.
2. Compatibility between different blockchain technologies, i.e. Bitcoin's UTXO model is not compatible with Ethereum's account model.
3. Consensus mechanisms lack flexibility because of different participants, the requirements of consensus mechanisms in public chains and permissioned chain are different.
4. The lack of consideration for business compliance requirements, i.e. identity and KYC/AML requirements of the financial industry which the current blockchain implementations cannot fully support.
5. The current blockchain implementations are not open to external actors. Most of the triggering criteria for smart contracts are set on the blockchain itself. There are no triggering criteria from off-chain data sources which can be used to build a connection with the real word.

Qtum proposes a series of innovations in blockchain technology and implementation, to offer a response to the above challenges. This includes a UTXO model based smart contract platform, a flexible consensus (Proof-of-Stake) mechanism for public and permissioned chains, the Master Contract concept, identity management through smart contracts.

1.3 Qtum Design Principle

1.3.1 Qtum Compatibility Design

- **Compatibility with Bitcoin and Ethereum network**

The Bitcoin Network is the largest blockchain ecosystem so far. Based on the network effect and Matthew Effect, the bitcoin ecosystem can be further expanded.

For Qtum's design architecture, we need it to remain compatible with Bitcoin's system, i.e., the UTXO transaction model. It allows Qtum to be compatible with the BIP protocol. In later stages, Qtum can accept more and more BIP's, i.e. lighting network, sidechains, drivechains and Zero Knowledge Proofs (Zcash) features and protocols, etc.

Ethereum was the first to change smart contracts from an idea to a reality, which further advanced the limit of Blockchain technology. Though the *Ethereum Virtual Machine* (EVM) has room for improvement, i.e. Transaction-Ordering Dependence Attack/Timestamp Dependence Attack, Mishandled Exceptions, etc., it is still the only tested smart contract virtual machine. Therefore, to be compatible with EVM is very important. Qtum is designed to compatible with EVM so that most smart contracts on Ethereum can also be ported to Qtum.

- **Backwards Compatibility**

It is very important for a system to be backwards compatible. The smart contracts created on older versions of the software should also operate well on a new version. The system won't require users to upgrade. This will bring more convenience to users. Once a smart contract is deployed it won't have to be changed forever if the system is backwards compatible, this will help avoid potential problems for already deployed smart contracts. Similar problems happened between EVM 2.0 and EVM 1.0. The designer of a blockchain system needs to consider such problems.

1.3.2 Qtum Module Design Approach

A modular design approach can help developers better maintain the system. In Qtum, we designed the following three modules:

- **Qtum Tech Module**: Qtum Core, Qtum VM, Qtum Identity, Qtum Storage, etc.
- **Qtum UI Module**: Qtum IDE, Qtum Mobile, Qtum Web, Qtum Node
- **Qtum Business Module**: Qtum Financial, Qtum legal and risk, Qtum Industry

1.3.3 Qtum Security Principle

Reliability of Qtum infrastructure technology
The first stage of Qtum is to provide a UTXO based PoS smart contract platform, which can be applied to different industries. The UTXO model is key to the Bitcoin network. The code is mature and well tested. By being compatible with the UTXO model, Qtum can attract more Bitcoin developers and use more tools. It has also been proved to be more secure than the account based model.

For the consensus part, apart from Proof-of-Work (including Sha256/sha3/scrypt/X11/X13, etc.) introduced by Adam Back and implemented by Satoshi in the Bitcoin network and other developers of other networks, Proof-of-Stake is another widely used protocol. In the evolution of PoS protocols, various potential attacks (i.e., Coin age attack, PoS nodes offline assault, pre-calculated Hash attack, etc.) have been remediated.

Therefore, in the Qtum system, we will use the PoS protocol as our consensus base. We will name it IPoS (Incentive Proof-of-Stake) as we will add an incentive mechanism. Currently, we still use the PoS 3.0 protocol in the Qtum test network. At a later stage, we will transfer to the newly designed IPoS protocol.

Qtum’s current test network supports EVM because this is the only tested smart contract virtual machine so far. Later, we will focus on 1) developing a more strict smart contract coding language; 2) supporting an enhanced EVM; 3) the possible integration of other virtual machines (LLVM, Lua, NodeJS).

Qtum Platform security strategy
Qtum’s platform will undergo strict testing before deployment; this includes software functionality testing, P2P network performance testing, potential attack tests, reliability tests, security and coding audits, etc.

The Qtum developers will release each update on the test network, which will allow public scrutiny of the code before a live release.

1.3.4 Qtum Usability Strategy

There will be different versions of the Qtum wallet, aimed at delivering multiple levels of functionality, based on the needs of the user. This means a general wallet will be available to meet the needs of most users; there will not be expert level debugging features, but all of the basic send and receive functionality will exist. For developers and administrators, we expect they will compile the source code and operate the daemon in the same way they do for Bitcoin

Core. Despite this, Qtum will release a pre-compiled binary for advanced users on different operating systems. These will include all of the functionality that could be expected from a fresh compile, but aimed at users that may not have the technical skills needed to compile Qtum Core.

Similar to Bitcoin Core, there will be an API service based on Standard JSON-RPC. In addition, we will also provide an Integrated Development Environment, for debugging and development.

Users may also access DAPP services through a browser (Chrome or Firefox, etc.) by inputting Qtum://DappName to visit various DAPP’s. For example, users wanting to order (and pay for) a taxi can visit a decentralized DAPP, calltaxi, via inputting Qtum://calltaxi

Part 2 Qtum Implementation

2.1 Qtum Blockchain

One of the primary goals of Qtum is to build the first UTXO based Smart Contract platforms with a POS consensus model. This platform will be compatible with the Bitcoin and Ethereum ecosystems.

Qtum's target is to produce a variation of Bitcoin with EVM compatibility (Ethereum Virtual Machine). Through a practical design, Qtum hopes to push its industry use cases with their "Go-Mobile" strategy; this can also help Qtum promote blockchain technology to a wide array of internet users.

2.2 UTXO vs. Account Model

2.2.1 UTXO Model

In the UTXO model there are transactions inputs, bitcoins destroyed, and transactions outputs, bitcoins created by a transaction. 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 signing the private key created by the public address of the receiver (the new owner). In the Bitcoin network, miners generate bitcoins with a process called a Coinbase transaction, which doesn't contain any inputs.

Bitcoin uses a scripting language for transactions. In the Bitcoin network, the scripting system processes data by stacks (Main Stack and Alt Stack), which is an abstract data type with the LIFO feature: Last-In, First-Out.

In the Bitcoin client, the developers use Standard() function to summarize the scripting types, Bitcoin clients support: P2PKH (Pay to Public key Hash), P2PK (Pay to Publickey), MultiSignature (less than 15 private key signatures), P2SH (Pay to Script Hash), and OP_Return. By these five standard scripting types, Bitcoin clients can process complex payment logic. Besides that, a non-standard script can be created and executed, only on the condition that there must be a miner who will encapsulate non-standard transactions.

For example – P2PKH (Pay to Publickey hash) to explain the process of script creation and execution. Let's say we need to pay 0.01BTC to a bread shop to buy some bread and the bitcoin address of the shop is "Bread Address".
So, the output of this transaction is:
OP_DUP OP_HASH160 <Bread Public Key Hash> OP_EQUAL OP_CHECKSIG
The unlock script according to the lock script is:

```
<Bread Signature> <Bread Public Key>
```

The combined script with the above two:

```
<Bread Signature> <Bread Public Key> OP_DUP OP_HASH160
<Bread Public Key Hash> OP_EQUAL OP_CHECKSIG
```

Only when the unlock script and the lock script have the matching predefined condition, the execution of the script combination is true. It means the Bread Signature must be a signature signed by matching the private key of Bread Address, which is the valid signature of Bread Address, and then the result will be true.

Even though the scripting language of Bitcoin contains many characters, it is not Turing-complete. There is no loop function that means the volume of execution of transactions is limited, and it also means the complicity of transactions is limited. This scripting language is not a commonly used programming language. Those limitations mitigate the security risks by preventing complex payment conditions from happening, those that generate infinite loops or other complicated logic loopholes.

In the UTXO model, we can trace back the history of each transaction through the public ledger and is totally transparent. The UTXO model has the parallel processing capability to initialize transactions among multiple addresses indicating the extendibility. Last but not least, the UTXO model has a certain level of privacy. Users can use Change Address as the output of a UTXO. Nevertheless, UTXOs have no status, and Qtum's target is to implement smart contracts based on the UTXO model's innovative design.

2.2.2 Account model

Versus the UTXO model, Ethereum is an account based system. In the white paper of Ethereum, there is elaboration about the account model shown below:

"In Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields:

- The nonce, a counter used to make sure each transaction can only be processed once
- The account's current ether balance
- The account's contract code, if present
- The account's storage (empty by default)

"Ether" is the main internal crypto-fuel of Ethereum, and is used to pay transaction fees. In general, there are two types of accounts: externally owned accounts, controlled by private keys, and contract accounts, controlled by their contract code. An externally owned account has no code, and one can send messages from an externally owned account by creating and signing a transaction; in a contract account, every time the contract account receives a message its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn."

In Ethereum's system, account balances are managed in the account system, in which any increase or decrease of an account balance is illustrated more like a bank account in the real world, and every new block generated will possibly make an influence on the global status of accounts. Every account has its own balance, storage and code space base on that the contract is able to call accounts or addresses, and store the execution results accordingly into the storage.

In the current Ethereum account system, users can make one-to-one transactions via client/rpc, which means a transaction can only be made from one account to another for each time. Although it's possible to send to more accounts via smart contract, these internal transactions can only be revealed in the balance of each account, and it is difficult to track them on Ethereum's public ledger.

The UTXO model of the Bitcoin network, which ensures the consistency and traceability of bitcoin transactions, is the core design of Bitcoin. **Based on the network effect and the advantages of the UTXO model we learned from Bitcoin, we decided to choose UTXO model as the first step of Qtum.**

2.3 Consensus

Based on the technical requirements of reliability and decentralization, Proof-of-Stake 3.0 was selected as the consensus platform for Qtum's Blockchain.

There have been many discussions about consensus and which platform meets the needs of a particular project. The topics most widely discussed are: Proof-of-Work, Proof-of-Stake, Dynamic Proof-of-Stake, and Byzantine Fault Tolerance as discussed by HyperLedger. The nature of consensus is about how to achieve data consistency by running an algorithm in a distributed system. All of the discussions about consensus will definitively go back to ask the original question – how do we maintain the consistency of a distributed system? There are many opinions in this field, for instance, the Fischer Lynch and Paterson theorem which states consensus cannot be reached without 100% agreement amongst nodes.

In the Bitcoin network, miners participate in the network verification process by hash collision through Proof-of-Work. When the hash value of a miner is able to calculate and meet a certain condition, the miner could claim to the network that a new block has been mined. That is

$$\mathrm{Hash}(\mathrm{B\_Header}) \leq \frac{M}{D}$$

the Hash() represents to compute SHA256 by power of 2 times, with value range [0, M], and D is an integer between [1, M]. The SHA256 algorithm used by Bitcoin enables every node to verify each block quickly. The 80 byte BlockHeader varies with each different Nonce. The overall difficulty level of mining will be adjusted dynamically according to the total hash power of the Blockchain's network. When two or more miners solve a block at the same time, a small fork in the network happens. This is the point where the Blockchain needs to make a decision as to which block it should accept, and which one it should reject. In the Bitcoin network, the chain that has the most proven work attached to it, is selected as legitimate.

There are different Proof-of-Work 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.

Until now, most of Proof-of-Stake Blockchains can source their heritage back to PeerCoin, which was developed from an earlier version of Bitcoin Core.

Qtum Core chose Proof-of-Stake for basic consensus, but we will develop and deploy POS based on the latest Bitcoin source code.

In a traditional Proof-of-Stake transaction, the generation of a new block must meet the following conditions:

$$\mathrm{ProofHash} < \mathrm{coins} \cdot \mathrm{age} \cdot \mathrm{target}$$

ProofHash is computed by Stakemodifier, 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. The other problem caused by coin age is that nodes will be online intermittently after rewarding instead of being continuously online. Therefore, in the improved version of POS agreement, coin age is removed to encourage more nodes to be online simultaneously.

In the original Proof-of-Stake implementation, there are several security issues that may be possible due to coin age attacks, etc. The Qtum developers agree with the security analysis of the Blackcoin team and have worked to implement POS 3.0 into the latest Qtum Core. This should theoretically reward investors that 'stake' their coins longer, while giving no incentive at all to coin holders who leave their wallets offline.

2.4 Qtum Contract and EVM Integration

The Ethereum Virtual Machine is a stack-based virtual machine with a 256-bit machine word. Smart contracts which run on Ethereum use this virtual machine for their execution.
The EVM is designed for Ethereum's blockchain, and thus assumes that all value transfer will be done using an account-based method. Qtum is based on Bitcoin's blockchain design, however, and uses the UTXO-based model. Thus, Qtum has an **Account Abstraction Layer** which translates the UTXO-based model to an account-based interface for the EVM to use. As well as this there is an additional **Blockchain Interface** so that the EVM can directly access various details about the Qtum blockchain.

2.4.1 EVM Integration

All transactions in Qtum use the Bitcoin Scripting Language, just like Bitcoin. In Qtum however, there are 3 new opcodes.

1. OP_EXEC - This opcode will trigger special processing of a transaction (explained below) and will execute the EVM bytecode passed to it.
2. OP_EXEC_ASSIGN - This opcode will also trigger special processing like OP_EXEC. This opcode is passed a contract address and data to give the contract. It will then execute the contract's bytecode while passing in the given data (given as *CALLERDATA* in EVM). This opcode is also used for giving money to a smart contract.
3. OP_TXHASH - This opcode is used to reconcile an odd part of the accounting abstraction layer. It simply pushes the transaction ID hash of the current transaction being executed.

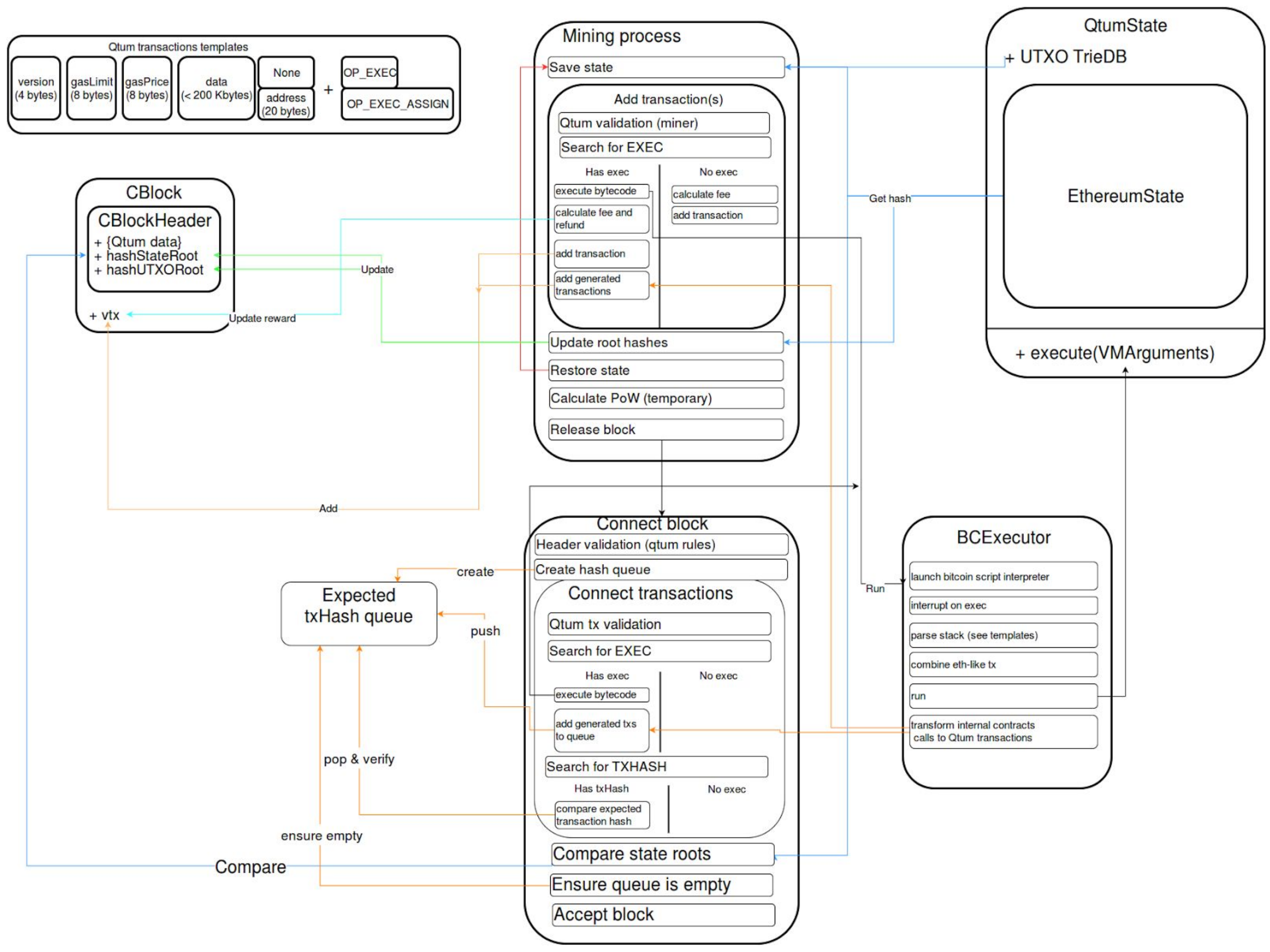

Diagram 1 : Qtum transaction processing diagram

Traditionally, scripts are only executed when an output is attempted to be spent. For example, with a standard public key hash transaction, though the script will be on the blockchain, it will not be either validated or executed in any way. Execution and validation does not happen until a transaction input references that output. At this point, if the input script (*ScriptSig*) does not provide a valid data to the output script that causes it to return 1, the transaction will not be valid.

Qtum however, must accommodate smart contracts which execute immediately when merged into the blockchain. It does this by special processing of transaction output scripts (ScriptPubKey) which contain either OP_EXEC or OP_EXEC_ASSIGN. When one of these opcodes are 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 more as strictly a way to carry data to the EVM. When the EVM is executed by either of these opcodes, the EVM can change state within it's own state database, exactly like a similar contract being executed on Ethereum.

For Qtum smart contracts to be as easy to use as possible, we have to authenticate the data sent to a smart contract as well as its creator as coming from a particular pubkeyhash address.

In order to prevent the UTXO set of the Qtum 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.

2.4.2 Qtum Account Abstraction Layer （Qtum AAL）

The Ethereum Virtual Machine is designed to function on an account-based blockchain. Qtum however, being based on bitcoin, uses a UTXO-based blockchain. To handle this, Qtum contains an Account Abstraction Layer which will allow the Ethereum Virtual Machine to function on the Qtum blockchain without significant modifications to the virtual machine nor existing Ethereum contracts.

The EVM account model exposed to smart contract programmers is fairly simple. There are operations that can check the balance of the current contract and other contracts on the blockchain, and there are operations which can send money (attached to data) to other contracts. Although these actions seem fairly basic and minimalistic, they are not trivial to do within the UTXO-based Qtum blockchain. Thus, the Account Abstract Layer's implementation of these operations may be more complex than expected.

First off, when a smart contract is deployed to the Qtum blockchain it is assigned and callable by its transaction hash. A newly deployed contract's balance will also be zero. There is currently no protocol in Qtum that allows a contract to be deployed with a non-zero balance. In order to send funds to a contract, a transaction will be created which uses the OP_EXEC_ASSIGN opcode.

The output script which sends money to the contract would look similar to this:

```
1; the version of the VM
10000; gas limit for the transaction
100; gas price in Qtum satoshis
0xF012; data to send the contract (usually using the Solidity ABI)
0x1452b22265803b201ac1f8bb25840cb70afe3303; ripemd-160 hash of contract txid
OP_EXEC_ASSIGN
```

This transaction script is fairly simple and hands off most of the transaction processing to the OP_EXEC_ASSIGN opcode. The actual value amount given to the contract from this (assuming there is not an *out-of-gas* condition) is `OutputValue - GasLimit.` The exact details of the gas mechanism will be discussed later. When this output is added to the blockchain, it becomes an output that *belongs* to the contract's account. This output's value will be reflected in the balance of the contract.The balance is simply the sum of the outputs which are spendable by the contract.

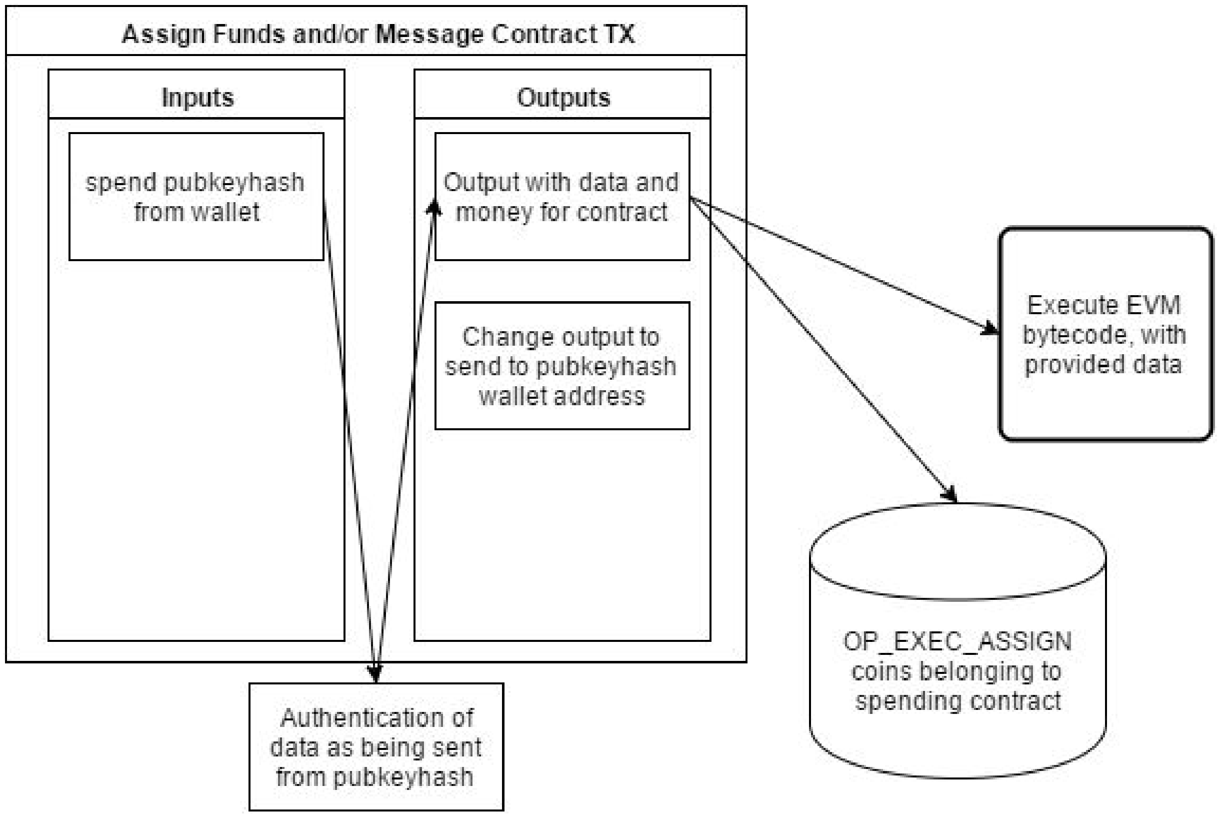
Assign Funds and/or Message Contract TX
Inputs
Outputs
spend pubkeyhash from wallet
Output with data and money for contract
Change output to send to pubkeyhash wallet address
Execute EVM bytecode, with provided data
OP_EXEC_ASSIGN coins belonging to spending contract
Authentication of data as being sent from pubkeyhash

Diagram 2 : Assign Funds and/or Message contract TX

Although this diagram shows sending funds to a contract from a standard public key hash output, the method for sending money from one contract to another is nearly identical. It is also possible to send funds from P2SH and non-standard transactions to a contract.

When the contract wishes to send funds to another contract or public key hash address, it *spends* one of its owned output. The mechanism by which it does this involves what will be called **Expected Contract Transactions**. These transactions are special in that they must exist in a block in order for the block to be considered valid by the Qtum 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.

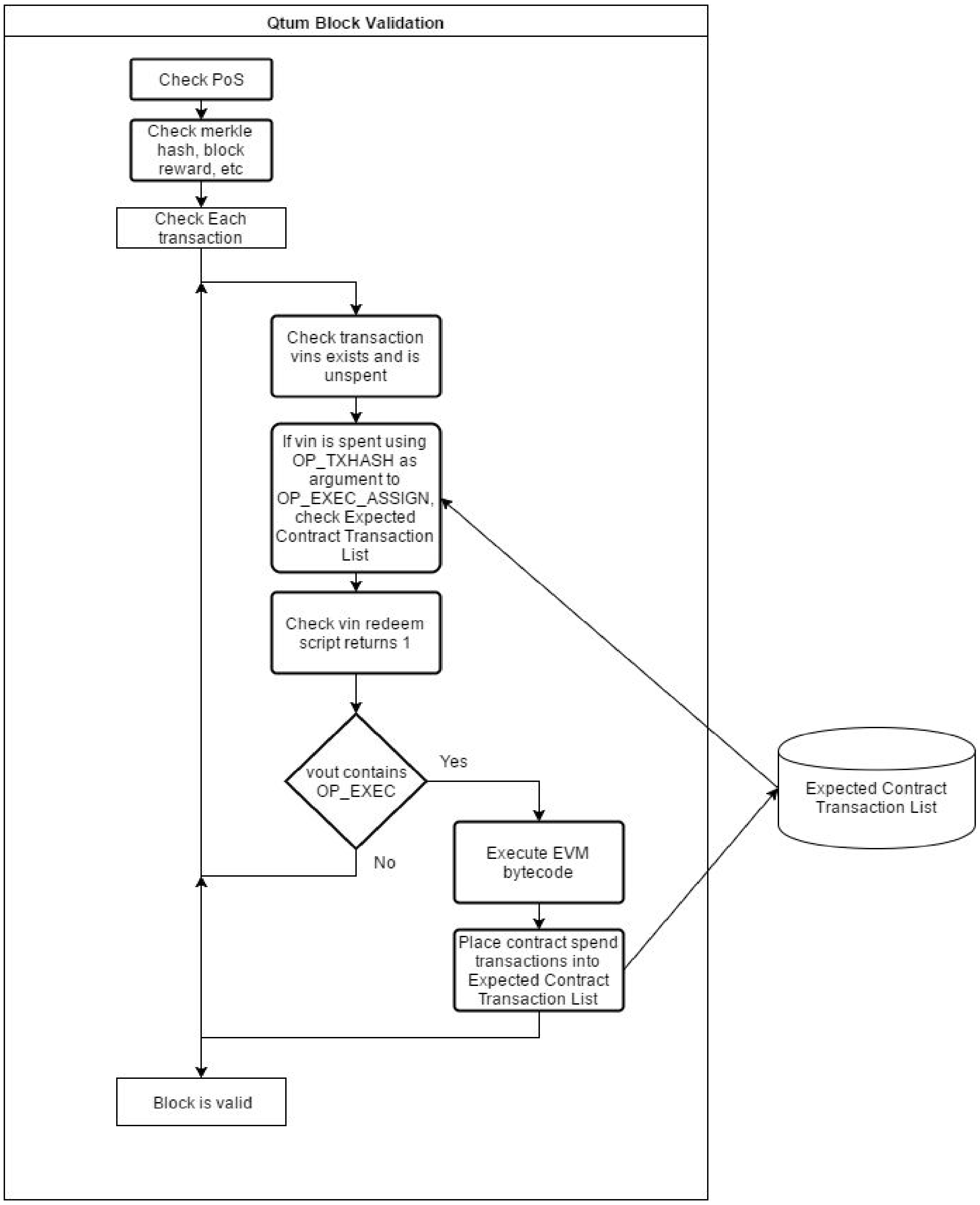
Qtum Block Validation
Check PoS
Check merkle hash, block reward, etc
Check Each transaction
Check transaction vins exists and is unspent
If vin is spent using OP_TXHASH as argument to OP_EXEC_ASSIGN, check Expected Contract Transaction List
Check vin redeem script returns 1
vout contains OP_EXEC
Yes
No
Execute EVM bytecode
Place contract spend transactions into Expected Contract Transaction List
Expected Contract Transaction List
Block is valid

Diagram 3: Qtum Block Validation showing Expected Contract Transaction List

The primary mechanism that makes Expected Contract Transactions work is the new opcode, OP_TXHASH. Internally, both OP_EXEC and OP_EXEC_ASSIGN have two different *modes*. When they are executed as part of the output script processing, the EVM is executed. When they are executed as part of input script processing, however, the EVM is not executed (as this would result in double execution). Instead, the OP_EXEC and OP_EXEC_ASSIGN opcodes will behave mostly like no-ops. They will return either 1 or 0 (spendable or not spendable, respectively) based on the transaction hash given to them. This is why OP_TXHASH is so important to the functioning of this concept. The OP_EXEC and OP_EXEC_ASSIGN opcodes will check the Expected Contract Transaction List when they are in a state of attempting to be spent. If the transaction hash passed in (from OP_TXHASH normally) to them exists in the Expected Contract Transaction List, the result will be 1, or spendable. Otherwise, it will return 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 should be spendable (i.e., when the contract tries to send money somewhere). This logic is somewhat circular but results in a secure and sound way of allowing a contract's funds to be spent only by that contract, and to behave mostly like a normal UTXO transaction.

One problem not yet touched on is that if a contract has more than one output that can be spent, each node could pick different outputs, and thus completely different transactions for spending OP_EXEC_ASSIGN transactions. This is resolved in Qtum by a consensus-critical coin picking algorithm. This coin picking algorithm is similar to the standard coin picking algorithm used within a user's wallet. However, it has been greatly simplified to avoid the risk of DoS attack vectors and so that the consensus rules can be as simple as possible. With this consensus-critical coin picking algorithm, there is now no possibility of different nodes picking different coins to be spent by a contract. Any miner/node who picks different outputs would fork away from the main Qtum network, and their blocks would not be valid.

To put all of this together, when an EVM contract sends money either to a pubkeyhash address, or to another contract, this will cause a new transaction the be constructed. The consensus-critical coin picking algorithm would choose the best outputs out of the contract's pool of owned outputs. These outputs would then be spent as inputs with the input script (ScriptSig) consisting of a single OP_TXHASH opcode. The outputs would thus be 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's hash would be added to the Expected Contract Transaction List, and then the transaction itself would be added to the block immediately after the contract execution transaction. Later, when this constructed transaction is validated and executed, the Expected Contract Transaction List will be checked, confirmed to be correct, and then this transaction's hash will be removed from the Expected Contract Transaction List. Using this model, there is no way of spoofing transactions to make them spendable by providing a hardcoded hash as the input script, instead of using OP_TXHASH.

This abstraction layer makes it so that EVM contracts can be oblivious to coin picking, and specific outputs, and can instead know only that they and other contracts have a balance, and money can be sent to these contracts as well as outside of the contract system to pubkeyhash addresses. With this in place, contract compatibility between Qtum and Ethereum should be very strong, and very few modifications (if any) will need to be done to port an Ethereum contract to the Qtum blockchain.

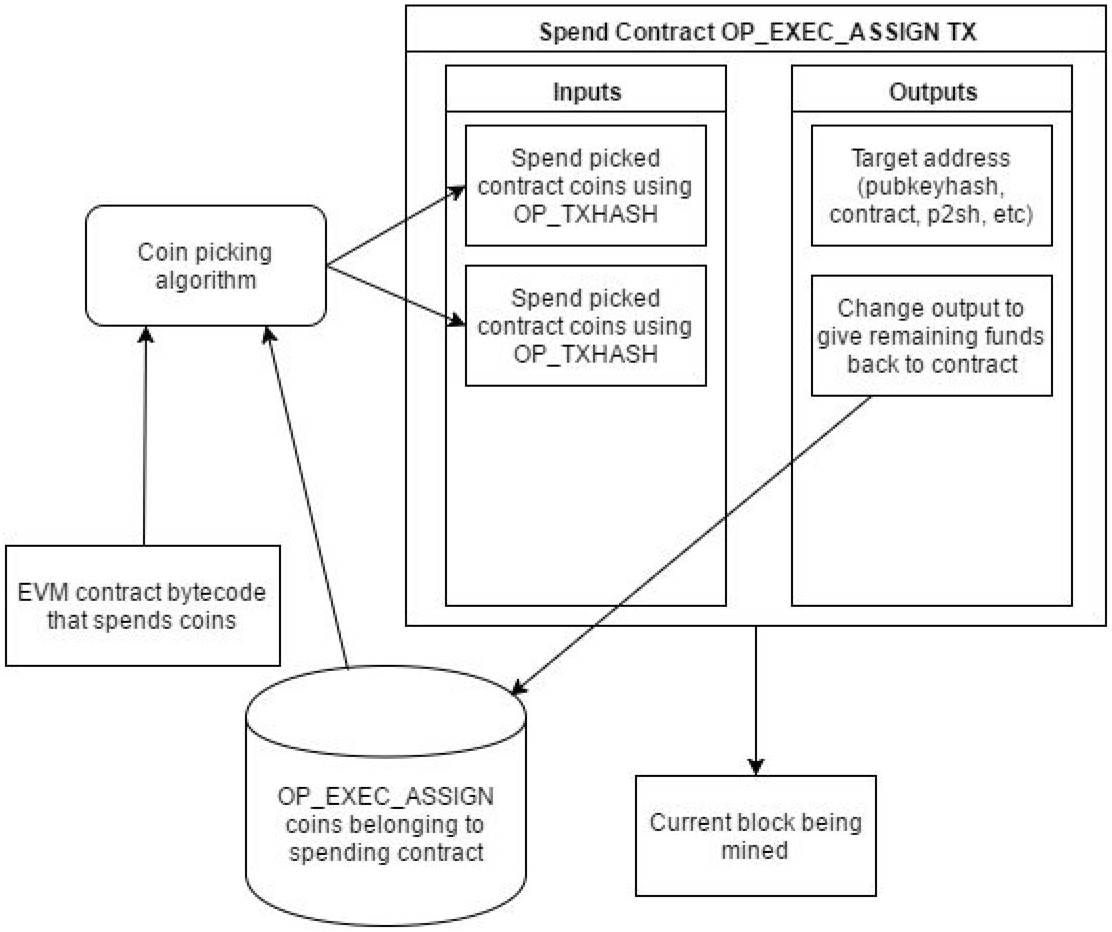
Spend Contract OP_EXEC_ASSIGN TX
Inputs
Spend picked contract coins using OP_TXHASH
Spend picked contract coins using OP_TXHASH
Outputs
Target address (pubkeyhash, contract, p2sh, etc)
Change output to give remaining funds back to contract
Coin picking algorithm
EVM contract bytecode that spends coins
OP_EXEC_ASSIGN coins belonging to spending contract
Current block being mined

Diagram 4 : Spend contract OP_EXEC_ASSIGN transaction

2.4.3 Added Standard Transaction Types

The following are the standard transaction types which were added to Qtum. They are documented here as Bitcoin Script templates:

Deploying a new contract to the blockchain should use an output script which looks like so:

```
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:

```
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 type which can spend either of these. This is because they can only be spent by using the Expected Contract Transaction List, and thus these spending transactions would not be broadcast nor valid on the P2P network.

2.4.4 Gas Model

One major problem Qtum faces with adding Turing-completeness to the Bitcoin blockchain is that it is no longer reasonable to rely on only the size of a transaction to determine the appropriate fee paid to miners. This is because a very simple and small transaction could implement an infinite loop and bring the entire blockchain to a halt while miners attempt to process it. Thus, the Qtum Project has ported the concept of *gas* from Ethereum. The gas concept can be summarized by saying that each EVM opcode executed has a price, and each transaction has an amount of *gas* which can be spent. Whatever amount of gas remains after the transaction is complete will be refunded back to the sender. Also if the amount of gas required to execute a contract exceeds the amount of gas available to a transaction, then the transaction's actions and state changes are reverted. This means any permanent storage that has been modified will be reverted to its original state, and any spending of contract funds will be reverted so that they are not spent. Even though all of this state is reverted, all of the gas of the transaction is consumed and given to the miner processing it. This is because the computing resources have already been spent by it's processing, so even though it's not safe to cause any state changes on the blockchain, the processing power has been spent and should go to the miner for its effort.

Although Qtum uses the gas model from Ethereum, it is expected that the *gas schedule* (gas price of each EVM opcode) will significantly differ from Ethereum. This is because in Qtum some operations are more expensive than in Ethereum, and some operations are cheaper. The exact values will be determined by looking at existing prices in Ethereum and comparing the amount of processing and blockchain resources required for each opcode in comparison to Qtum.

When creating a contract funding or deployment transaction, the user specifies two specific items for gas. The first is the *GasLimit*, which is how much gas can be consumed by this contract execution. The second is the *GasPrice*, which is the exact price of each unit of gas in Qtum satoshis. The maximum Qtum expenditure of a contract execution can thus be easily computed by *GasLimit* multiplied by *GasPrice*. If this maximum expenditure exceeds the transaction fee provided by the transaction, then the transaction is considered invalid and will not be mined or processed. The remaining transaction fee after this maximum expenditure is subtracted is the *Transaction Size Fee*. This is analogous to the standard Bitcoin fee model. To determine the appropriate priority of a transaction, miners must now look at two variables. First, the transaction size fee should be appropriate for the total size of the transaction (usually determined by a minimum amount of *coins per kilobyte* formula). The second variable is, of course, the *GasPrice* of a contract execution. Together, proof-of-stake miners have a great degree of choice in choosing the most important and profitable transactions to process and include in a block. This allows the fee model to work like a free market, with miners and users optimizing for the best fee that suites their transaction's speed and the price they are willing to pay.

(a) Refunds

Using the UTXO model, funds sent to miners as transaction fees are non-negotiable. There is no way for a miner to partially refund the fee if the transaction was easier for the miner to process than expected. However, for the gas model to be useful, there must be some method to refund some of the funds back to the sender. Moreover, there must be a way to roll back the state of a transaction which runs out of gas, but also a way to give its gas fees to the miners.

Refunding gas fees in Qtum is made possible by creating new outputs as part of the miner's coinbase transaction. A new block validation consensus rule was added as well, to ensure that these refunding outputs are required to exist in the coinbase transaction. Otherwise, miners could choose to not refund these funds.

The sender of a transaction's funds for refund purposes is considered to be the first input's referenced output. The refund is given back to it by simply copying the output script.Currently, for security reasons, this script can only be a standard pay-to-pubkeyhash or pay-to-scripthash script. This restriction may be lifted later if it is determined to be safe to do so.

For reference, the OP_EXEC_ASSIGN has this format (for assigning funds to a contract):

Inputs: (in push order)

- Transaction hash for spending [optional]
- version number (VM version to use, currently just 1)
- gas limit (maximum amount of gas that can be used by this exec)
- gas price (How much qtum each gas unit is)
- data (data to be passed to this smart contract)
- smart contract address

Outputs: (in pop order)

- Spendable (if the funds are currently spendable)

So, an example EXEC_ASSIGN might look like this:

1

10000

100

0xABCD1234...

3d655b14393b55a4dec8ba043bb286afa96af485

```
EXEC_ASSIGN
```

And if the VM execution results in an out of gas exception, this vout will be spent (by the next transaction in the block) using this redeem script:

```
OP_TXHASH
```

And the generate vout for this transaction will be a pubkeyhash script taken from the `vin[0].prevout` script. In this early version of Qtum, only pubkeyhash senders are allowed for VM funding transactions. Although other forms can be accepted into blocks and will result in VM execution, the `msg.sender` in the EVM will be "0", and any out of gas or gas refund needed will result in the contract getting to keep these funds

(b) Partial Refund Model

For the other side of the gas model, it is also necessary to refund the unspent portion of the gas. This is so that people can commit to spending a large amount of funds to ensure their contract is executed properly, but what gas they didn't use they get back as a qtum refund.

The return address for gas is expressed on the blockchain as an the `vin[0].prevout` script of the sending transaction. Gas is sent to a contract by using the standard bitcoin transaction fee mechanism. So, the new fee model slightly augments this to make the transaction fee:

```
gas_fee = gas_limit * gas_price

txfee = vin - vout + gas_fee + tx_relay_fee

refund = txfee - used_gas - tx_relay_fee
```

Note that there is a proposal for making it so that miners can evaluate both the tx_relay_fee and the gas_price under a single "credit_price" value to determine transaction priority.

As the contract is executed, gas tokens are subtracted from the total fee (by multiplying by `gas_price`). After the contract's execution has completed, the remainder of this gas_fee must be returned to the given `gas return script`. This should be accomplished by adding an output to the coinbase transaction (what the miners use to retrieve their block reward). The vout that is added to the coinbase is a pubkeyhash from `vin[0].prevout`. In order to receive a gas refund, this must be a pubkeyhash vout that is spent. Otherwise, the gas refund will remain with the miner (and in out of gas condition, the funds sent will remain with the contract).

Note that it is currently only possible to have one EVM contract execution per transaction, so there should never be a case arise where two contract executions attempt to share the transaction fee. This may be enabled at a later point when a few other problems are solved with multiple EVM executions per transaction.

(c) Important GAS edge cases:

Miners must be careful about contracts gas/fund return scripts. If the gas return script output will cause the block to exceed the maximum block size, then the contract transaction can not be put into this block, and its execution must take place again in the next block to mine instead. Miners should always ensure there is enough room left in the candidate block for the gas return script before attempting to execute the contract. Not following this rule can result in a contract needing to be executed more than once after finding that the refund script won't fit into the current block.

If there are no gas funds to return, no vout will be made for returning the funds.

It is *consensus-critical* that the transaction fee include the `gas_fee`. If a transaction is attempted to be added to a block which would result in a negative gas refund, or the `gas_fee` is less than the `txfee`, then the transaction is invalid; thus any block with such a transaction will also be invalid.

No transaction output script is valid that has more than 1 OP_EXEC or OP_EXEC_ASSIGN opcode. This limits scripting abilities, but is much easier than dealing with the recursion/multiple execution problems. In this way, static analysis can easily determine if the script is invalid, rather than needing to execute the script to determine it.

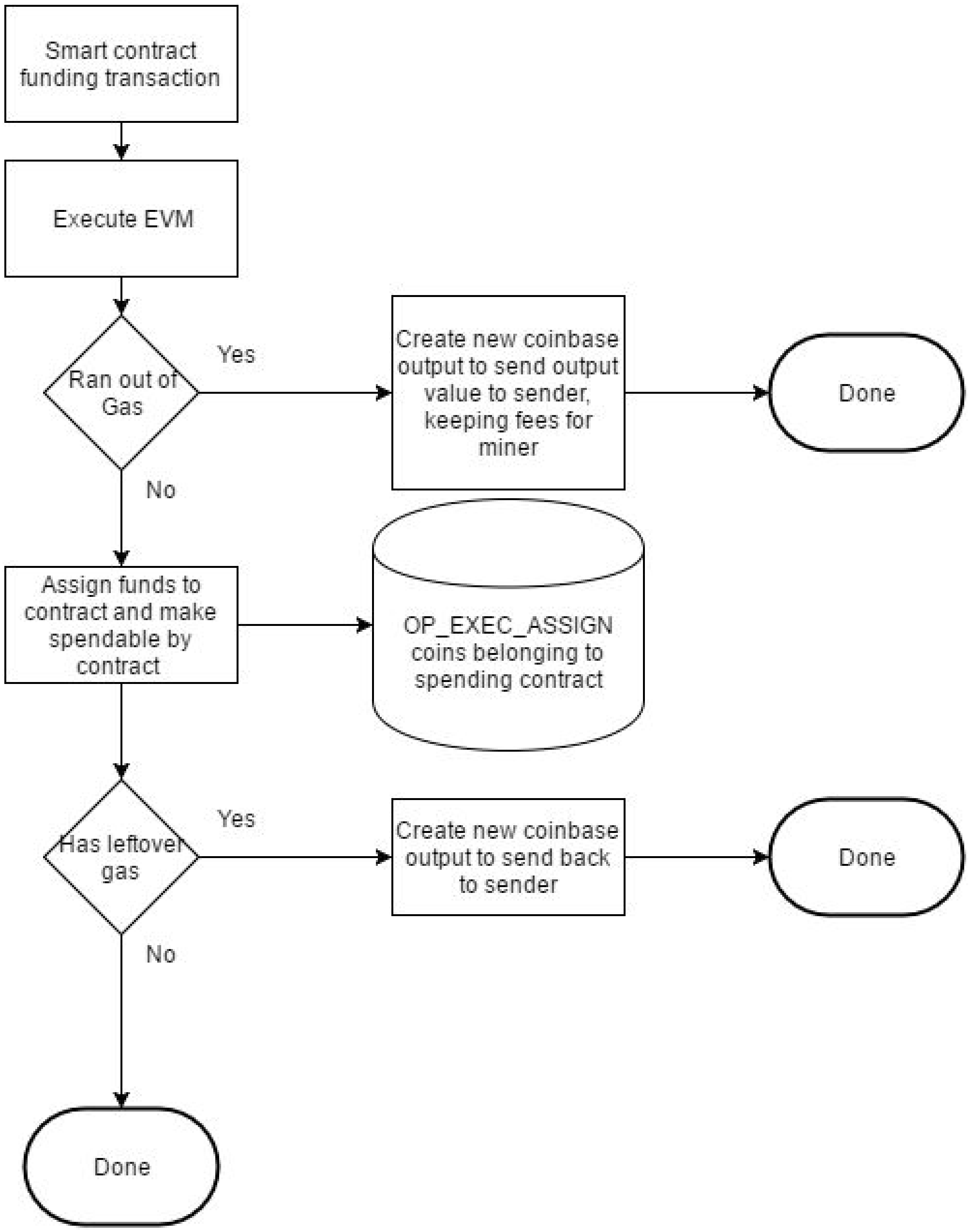
Smart contract funding transaction
Execute EVM
Ran out of Gas
Yes
Create new coinbase output to send output value to sender, keeping fees for miner
Done
No
Assign funds to contract and make spendable by contract
OP_EXEC_ASSIGN coins belonging to spending contract
Has leftover gas
Yes
Create new coinbase output to send back to sender
Done
No
Done

Diagram 5: Gas Refund Model

Part 3: Qtum Applications

3.1 DAPP Store

The Qtum system is devoted to support DAPPs on a technical level, through the introduction of Go Mobile. This will turn DAPP ideas into products and allow the average internet user to understand the value of blockchain technology.

DAPPs facing different industries could bring blockchain technology to clients and industries. Possible DAPPs could include social networks, storage, DNS services, etc.

Blockchain technologies provide the fundamental structure for constructing Decentralized Applications. In Qtum, developer preparation work is simplified by completing the Qtum API design.

3.2 Industry Oriented

The Qtum system supports the application needs of multiple industries: such as the financial industry, , social networking, gaming, digital assets, etc. Qtum smart contracts can provide support for more complex business logic via a Turing-complete programming language.

3.3 Identity and Privacy

Qtum will provide an optional identity module which is essential to integrate with various industries. The Qtum system will manage users' identities through smart contracts.

The developers of Qtum will develop smart contracts based on identity and share the source code with a third party. With the involvement a third-party credit agency, the verified customers will have more priority in Qtum.

Since the Qtum system is compatible with the UTXO model we can integrate with various privacy protocols.

3.4 Oracles and Data Feeds

In Qtum, oracles represent trustworthy data sources, entities, nodes, and public addresses. Oracles fulfill their responsibility through supply trusted data. Third parties can profit by offering data as a service to interested parties.

Usually, the trustworthy data sources will come from institutions with public reputation such as weather reporting networks, or the results of a sporting match. Developers could use APIs from those institutions to obtain the data such as an HTTP request.

We will use game theory to design the data feed in Qtum by using different data sources for deposit as a guarantee and rewarding the most trustworthy data source.

3.5 Go Mobile

The “Go Mobile” strategy is in the mind of the Qtum development team, and it is also an important step to make blockchain technology set foot in the world. In the Qtum ecosystem, we do not only give full support for the mobile application strategy, but also work together with developers from third parties to provide mobile services for clients: including mobile wallets, Dapps, smart contract applications, etc. We encourage developers from third parties to join us push forward blockchain technology in China and develop a blockchain that can be used by common internet users. We will offer a Qtum mobile wallet and a mobile contract API.

Future Directions

Here is a list of the future development:

1. More VMs support
2. Privacy based on ZCash protocol
3. Affinity Spending
4. Enabling P2SH contract ownership/funding
5. Dynamic gas model
6. Lifting various restrictions (such as 1 contract exec per transaction)
7. Making contracts easier to use for consumers, such as a special address type which combines both the contract address and the data to be sent to the contract
8. GitHub but for verified smart contracts

References

[1] https://en.bitcoin.it/wiki/Category:History
[2] https://github.com/bitcoinbook/bitcoinbook
[3] https://www.bitcoin.org/bitcoin.pdf
[4] https://github.com/ethereum/wiki/wiki/White-Paper
[5] Arvind Narayanan et al., Bitcoin and Cryptocurrency Technologies, 2016, http://www.the-blockchain.com/docs/Princeton%20Bitcoin%20and%20Cryptocurrency%20Technologies%20Course.pdf
[6] N. Szabo, Smart contracts, 1994, http://szabo.best.vwh.net/smart.contracts.html
[7] Edward Felten, Game Theory and Bitcoin, 2013, https://freedom-to-tinker.com/blog/felten/game-theory-and-bitcoin/
[8] Litecoin, 2011, https://litecoin.info/
[9] Evan Duffield and Kyle Hagan, Darkcoin: Peer-To-Peer Crypto-currency with anonymous Blockchain Transactions and Improved Proof-of-Work System, 2014, https://www.dash.org/wp-content/uploads/2014/09/DarkcoinWhitepaper.pdf
[10] Evan Duffield and Daniel Diaz, Dash: A Privacy Centric Crypto Currency, 2015, https://www.dash.org/wp-content/uploads/2015/04/Dash-WhitepaperV1.pdf
[11] Sunny King and Scott Nadal, PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake, 2012, http://web.archive.org/web/20131228174819/http://peercoin.net/bin/peercoin-paper.pdf
[12] Novacoin, 2013, http://coinwiki.info/en/Novacoin
[13] Nxt, 2013, http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt