Neptune White Paper

Alan Szepieniec and Thorkil Værge

April 23, 2021

42 min read



Introduction

The combined market cap of cryptocurrency networks numbers in the trillions of dollars. The industry's rapid expansion from Bitcoin's humble beginnings attests to the market demand for an alternative to fiat money.

The horizontal expansion of this industry indicates more than just cryptocurrencies' increasing share of the money and the financial markets, it indicates that the innovators and market participants disagree about the most urgent problems to be solved by the blockchains on which cryptocurrencies exist. Three qualities at the center of this market debate are *privacy*, *programmability*, and *scalability*.

- **Privacy.** The default mechanism by which a blockchain enables transaction verification is by broadcasting the entire ledger. The verifier then traces the origin of the coins in question back to the point where they were minted. In the context of cryptocurrencies, privacy is the technology that obfuscates the ledger and thwarts traceability, while guaranteeing that the proper holders of the coins consent to the transactions that spend them and that no unintended inflation occurs.

Privacy is the flipside of fungibility; they protect cryptocurrency holders from being targeted by malicious third parties. Traceable transactions induce a legal risk on the part of the holder or recipient. For instance, perhaps the money can be traced back to illicit or unpopular activities, or perhaps the victim of a theft can trace the funds and demand their return to the rightful owner. Although traceability may sound like a positive feature, it is a poor quality for money to have as it confers an obligation unto the recipient to verify the funds' clean history, lest he be liable for enabling illicit activity. The threat of liability translates to a higher cost of transacting and holding currency -- whereas good money minimizes these costs. The engineering solution is to void the legal argument by ensuring plausible deniability at the protocol level. An emulation of this solution has a long-standing precedent in Western legal tradition (link).

- **Programmability.** The Bitcoin client accommodates joint ownership and control as well as transfers conditional on temporal constraints. However, its virtual machine language does not support loops or jumps by design.

  In contrast, the virtual machine language of Ethereum is Turing complete by design. Any algorithmically decidable spending policy can be encoded as an Ethereum smart contract, only constrained by a maximum number of instructions per smart contract invocation. This base layer innovation has led to a flurry of higher-level applications, starting from tokenization of shares and debt instruments, culminating in automatic market makers valued in the billions even mere months after deployment. Since there is no central single point of failure, at least in theory, these overlay protocols are characterized as *decentralized finance* (DeFi).

- **Scalability.** Scalability refers to the capacity of a network to support a large number of transactions and state transitions per time unit. As demand grows for a network with limited capacity, the price of operating on this system increases. The scalability question addresses this paradox: how to enable adoption, when that adoption is self-limiting.

The reason why blockchains have a limited throughput capacity is because there must be consensus about the information that exists on it. A naïve increase in the block size limit increases the workload of participants in the network, which ultimately reduces the number of full nodes and miners. This reduced participation a) centralizes the network making it more susceptible to attack and regulatory capture; and b) reduces the mining power consumed to secure the network against reorganizations. The resulting security degradation will drive high-value transactions to a blockchain that offers a matching level of security.

Layer 2 protocols address the scalability question by shifting perspective. Space on the blockchain is a precious resource and should be reserved for information important enough to justify the cost of consensus. The blockchain is not a means for distributing information; instead, it is better characterized as a reconciliation protocol to be used for settling disputes about transactions and transitions that happen off chain.

By moving most transactions off chain, one transaction no longer competes with another transaction for confirmation. The compromise is a stronger trust assumption: the users rely on the availability of a mechanism to dispute transactions when necessary. The pathway towards scalability consists in engineering a *robust dispute resolution framework*.

Neptune

Archived copy · cryp2nova.com

This white paper presents Neptune, a new Layer 1 blockchain protocol designed to support smart contracts with privacy and with scalability. Key features include:

- privacy within the UTXO-model of transactions
- smart contracts whose execution is made efficiently verifiable with SNARKs
- native compatibility with Layer 2 protocols
- fast synchronization using SNARKs
- small data-storage footprint using SNARKs
- proof-of-work with Nakamoto consensus and adaptive block size to make use of surplus network capacity
- post-quantum cryptography
- finite asymptotical limit on the money supply

The rest of this paper elaborates on some of these design considerations, and follow up with a technical overview that sketches how these properties are achieved.

Design Philosophy

Privacy

Zero-knowledge proof systems enable a prover to run a program and prove to the verifier that it was executed correctly, even while all or part of the input remains secret. It is a tool seemingly tailor-made for blockchains, where network participants need to verify protocol contributions and where users need to protect cryptographic key material or other sensitive information.

The privacy-friendly coins ZCash and Monero achieve privacy by using zero-knowledge proofs to obfuscate the origins of transactions. A transaction on these blockchains come with a list of coins and a zero-knowledge proof that *one* of them is being spent, but the proof hides exactly which one. A Monero transaction provides two decoy origins, whereas in ZCash the set of origins is exactly the set of all transactions in the history up until that point.

This decoy strategy is fundamentally limited. It is never possible to prune the unspent transaction output (UTXO) set, because spent transaction outputs are useless as decoys. The storage requirements of nodes that generate transactions and verify them will grow linearly with the total number of transactions in all of the currency's history.

Mimblewimble is a technology for private coins that offers a sounder strategy in this respect. There is a UTXO set, and instead of decoys, privacy is achieved through coinjoin and cut-through. The cryptography allows transactions to be merged (coinjoin), thus blurring the lines of ownership; and it allows the elimination of UTXOs that are simultaneously inputs and outputs in the same transaction (cut-through). In order to achieve this flexibility in the transaction format, the Mimblewimble protocol sacrifices the ability to transact non-interactively. A sender needs to run a multi-pass protocol with a receiver in order to complete a transaction.

By using succinct arguments of knowledge (SNARKs), the same flexibility can be achieved *without* sacrificing non-interactivity. Proof-carrying data (PCD) is the construction on top of these proof systems that allows network participants to generate SNARKs that attest to *a*) the integral computation of a protocol contribution, and *b*) the successful verification of a similar SNARK received from a third party. In a nutshell, any transaction comes with a SNARK proving its authenticity; and a coinjoin or cut-through operation comes with a corresponding operation on the component SNARKs.

An additional viable strategy for achieving privacy in the UTXO model charges the miners with randomizing and shuffling the UTXO set. The UTXOs are represented by *publicly re-randomizable cryptographic commitments*. In addition to performing the mixing and re-randomization, the miners produce a proof of correct shuffling. Receivers of transactions need to locate their UTXOs by attempting to unlock all randomized commitments in a block. To reduce wasted work, the receiver should only try-to-unlock when he receives a *notification of payment*, a message that is preferably transmitted off-chain but can be on-chain if need be.

ZCash and Monero have an important design difference: the former has opt-in privacy, whereas the latter has mandatory privacy. Since the privacy technology is expensive in terms of both time and memory, most users of ZCash end up using the plain, Bitcoin-like backbone of ZCash. The end result is a smaller anonymity set for those users that *do* use privacy. In contrast, Monero requires that everyone make private transactions. As a result, the anonymity set is the entire Monero community. We side with Monero's design choice: there should be no opt-out of privacy as this reduces fungibility.

Programmability

Ethereum paved the way towards a smart contract ecosystem. However, Ethereum's account model of transactions is incompatible with the private UTXO model sketched above. Rather than lifting the privacy technologies to the account model, we lift the programmability to the UTXO model -- without sacrificing the privacy protections. We choose to reuse the term UTXO but in a more general sense because it does not necessarily represent an amount of currency.

There are two criteria to be satisfied whenever an UTXO is spent.

- The **lock script** is responsible for ensuring the proper evolution of ownership. This script is satisfiable only by the owner of the coin, and only if the right criteria for transfer are met. A transaction is not valid unless the lock script of each input UTXO is satisfied. The `lock_script` is the analogue of Bitcoin's `scriptPubKey`, although neither it nor its satisfying witness appear on the blockchain.
- The **type script** is responsible for ensuring the correct evolution of the state for all tokens involved in the transaction. Every token type has a unique type script. The states of Neptune's native currency tokens are just numbers, and their type scripts ensure that there is no inflation except in the

Archived copy · cryp2nova.com

coinbase transaction and then only in accordance with the monetary policy.

This model of programmable UTXOs is different from smart contracts in Ethereum, which have no owner and whose state is public. The two models have already been shown to be functionally equivalent, with the UTXO model being more conducive to scalability. The architecture of Neptune suggests that it is also more compatible with privacy enhancing technologies.

The key to simulating Ethereum-style public smart contracts is a UTXO whose lock script can be triggered without secret key material. This *anyone-can-spend* lock script must require though that at least one referenced output UTXO have the same lock script. In this way the anyone-can-spend UTXO guarantees that *anyone* can invoke a state evolution that the type script allows.

In addition to the `lock_script` and `type_script` of UTXOs, a third type of script called **public scripts** needs to be satisfied in order for a transaction to be valid. These scripts are public (*i.e.*, they do appear in the clear on the blockchain and are not hidden behind cryptographic commitments) and provide an access point for users to leverage the blockchain as a broadcast medium. Public scripts are fields of transactions and not UTXOs, and since their validation is more expensive than simple proof-verification, miners are free to drop transactions from their pools based on their internal price calculation.

Neptune's model of programmable UTXOs is comparable to the *cell model* of Nervos CKB. The difference here is that in Nervos the lock script, type script, their arguments, and token state are provided explicitly, whereas Neptune only provides cryptographic commitments thereto. The successful validation of the scripts is attested to by a zk-SNARK. Only the `public_scripts` are public, and these can be used only for simple logic or data availability. An alternative to using `public_scripts` for data availability is by disseminating the information via an off-chain protocol; this strategy incurs smaller transactions fees but requires more design care to ensure that no information is lost.

At any given point in time, a certain number of UTXOs will be dangling because they have not yet been consumed by any transactions; these are *Spendable UTXOs*, in contrast to *Spent UTXOs*. (We appreciate the irony in regards to the letter "U". In our terminology, "UTXO" is not an acronym but a standalone word denoting an edge in a specific graph.)

Scalability

The Lightning Network provides the blueprint for achieving scalability in terms of transaction throughput. Transactions should take place off chain by default, with cryptographic commitments on the blockchain to enable dispute resolution. For more general smart contracts, the participants should store state data off-chain, and exchange data as necessary via an overlay protocol that does not burden the blockchain. The only information that sees the blockchain are the cryptographic commitments to this state, as well as any evidence needed for litigating a dispute.

Transaction throughput is not the only metric that relates to scalability. The workload of nodes on the network is another. Neptune leverages several more technologies to lighten this load.

SNARKs are efficiently verifiable proof systems. Neptune uses them in several places.

1. To establish that the (arbitrarily complex) lock and type scripts have satisfying arguments, and that transactions are valid more generally.
2. To establish that the miner performed a correct mixing of the output UTXOs.
3. To establish, within every block, that the block is valid and the tip of the longest chain. (Technically: that the indicated amount of accumulated proof-of-work is correct; in case of a fork, nodes choose the block where this field is largest.)

Nodes represent the set of all UTXOs concisely by a Merkle Mountain Range Accumulator in order to reduce storage space. Spendable UTXOs are circulated with an authentication path attesting to their membership in this structure. Every block contains the roots of the most up-to-date Merkle Mountain Range. This structure obviates the need to disseminate the entire Spendable UTXO set. As a result, *full nodes have a constant memory footprint*.

Long-Term Value

As far as money goes, all value is long-term value. This observation informs a number of positions.

- **Proof-of-Work.** The consensus mechanism is Nakamoto-style proof-of-work. The various proof-of-stake alternatives compromise on simplicity as well as security model and have not withstood the test of time. Alternatives based on proof-of-delay or proof-of-space benefit the faster miner out of proportion to his share of consumed energy, thus incentivizing centralization.
- **Post-Quantum Security.** All cryptographic primitives in Neptune offer post-quantum security. When large scale quantum computers are built, they will undermine the security of other blockchain projects, but not Neptune. The mere threat posed by quantum computers hampers the capacity of any blockchain that is not quantum-secure to store value.
- **Deflation.** The monetary policy follows that of Bitcoin: there will be periodic halvings and a hard upper limit to the total supply.
- **Fork Policy.** Hard forks undermine the immutability of the blockchain history as well as of the monetary policy. They set precedents for centralization and control. The only justification for an unanticipated hard fork is a cryptographic vulnerability or similar emergency.

Technical Overview

Cryptographic Primitives

Zero-Knowledge and Succinctly-Verifiable Proof Systems

Archived copy · cryp2nova.com

A proof system is a protocol between a prover and a verifier in which the prover convinces the verifier that a certain computational claim is correct. The claim can be represented by a program, an input, and an output; the claim then states that the output was obtained integrally from the input. Without losing much generality, proof systems can be made non-interactive: in this case the verifier has a fourth argument, which is a string of cryptographic data called the *proof*.

The naïve verifier ignores this proof and runs the program on the input to see if the generated output matches with what is claimed. Proof systems become interesting, both theoretically and practically speaking, when the verifier has access to fewer resources than the prover does. *SNARKs* are one subset of proof system where the verifier runs asymptotically faster than the naïve execution of the program. In another subset called *zero-knowledge* the verifier remains ignorant of secret inputs to the computation. When a proof satisfies both descriptions, we refer to the proof system as well as to the resulting proof as a zk-SNARK.

An early generation of SNARKs (and to date, still the one that produces the shortest proofs) requires a trusted setup to produce public parameters. Neptune dispenses with the trusted setup procedure altogether (at the cost of larger proofs). Such proof systems are called *transparent*.

Concretely, Neptune uses a variant of the STARK proof system, so named because it is a transparent SNARK. This particular proof system which works well for computations represented in the von Neumann architecture (as opposed to the circuit model). The only cryptographic ingredient required to instantiate this proof system is a hash function with collision resistance. For an idealized hash function, quantum attacks have been proven non-existent; for a concrete hash function with no known quantum attacks, the proof system is therefore plausibly secure against quantum attacks. By instantiating the proof system with an *arithmetization-oriented hash function*, one can prove the correct execution of a proof-verification; this construction is a *recursive SNARK* and the building block of *Proof-Carrying Data (PCD)*.

The program whose correct computation is proven is represented in a list of instructions in a dedicated machine language called *Triton*. The Triton VM is a stack machine with access to random-access memory, that reads its instructions from memory. In terms of arithmetic operations, Triton supports:

- native arithmetic operations modulo pp, along with arithmetization-oriented hashing modulo pp, where pp is the prime number of elements of the field over which the proof system is defined
- computing Legendre symbols of native field elements
- arithmetic modulo 2^{32}232 for words of $32$32 bits, as well as bit-fiddling using separate algebraic execution tables
- operations needed to support RLWE commitments (once again using separate tables)

Ring Learning-With-Errors

Learning with Errors (LWE) is a family of cryptographic hard problems believed to resist quantum attacks. Ring Learning-with-Errors (RLWE) is an instantiation of this hard problem in terms of elements of a polynomial quotient ring. In essence, the hardness assumption states that it is difficult to recover ss and ee from AA and $B = sA + e$B = sA + e, where AA is uniformly random, and where ss and ee are drawn from a distribution of "small" elements. Except for the (*small*) error term ee, the map $A \mapsto B$A ↦ B is linear -- and as a result, much of the intuition about group exponentiation and the cryptographic constructions that rely on it can be recycled.

Because of this somewhat homomorphic property, RLWE lies at the basis of several post-quantum key encapsulation mechanisms as well as signature schemes. Neptune, however, does not require either of these primitives; instead, it requires a *publicly re-randomizable commitment scheme*.

In a group where the discrete logarithm problem is hard, the pair (G, aG)(G, aG) is a binding commitment to aa. This commitment can be re-randomized by a third party who does not know aa, simply by multiplying both elements by bb. The pair (bG, abG)(bG, abG) can be opened in the same way: by revealing aa. Importantly, only individuals knowledgeable of aa or bb can link this pair back to the original commitment.

To lift this construction to the RLWE setting, simply make sure that $\{a, b\}${a, b} are appropriately *small*, and don't forget to add equally small error terms $\{e_0, e_1, e_2\}${e0, e1, e2}. The resulting commitment is $(G, aG + e_0)$(G, aG + e0), and its re-randomization $(bG + e_1, baG + be_0 + e_2)$(bG + e1, baG + be0 + e2). Commitments under this scheme are not cryptographically hiding by default, so aa must have enough entropy or else two commitments to it could be linked.

The UTXOs in Neptune are RLWE commitments to *at least* three things:

- a *state*, which can evolve as the UTXO is consumed and re-generated by transactions;
- a *lock script*, which determines who can spend the UTXO; and
- a *type script*, which determines how the state evolves.

(In fact, a UTXO can commit to multiple (state, type script) pairs.)

In the case of Neptune native currency, the state is a positive dyadic rational number, the lock script determines ownership by verifying a signature, and the type script guarantees that there is no inflation. Overlay protocols can assign any meaning to these committed objects.

The re-randomizability of the commitment scheme is an essential quality for privacy as it achieves de-linking the output UTXOs of confirmed transactions from the input UTXOs of new transactions. Specifically, to the outside observer (who has no access to the miner's randomness) the output and input UTXOs are unlinkable even though they decommit to the same or matching information.

Arithmetization-Oriented Hash Functions

Archived copy · cryp2nova.com

A hash function is *arithmetization-oriented* if its evaluation circuit has a concise description in terms of finite field operations. Traditional hash functions like SHA2-256 and Keccak do not have this feature; they optimize for a small hardware or software footprint instead.

Arithmetization-oriented hash function are useful in the context of proof systems, where finite field operations are much cheaper than arithmetic over non-field algebras -- including bit fiddling. As a result, a prover wanting to prove the correct evaluation of a hash function in the context of a zero-knowledge proof, minimizes his workload by using an arithmetization-oriented hash function for this purpose.

Merkle Mountain Range Accumulator

A *Merkle tree* is a data structure that cryptographically authenticates membership of data elements in a set. The value of every non-leaf node in the tree equals the hash of the concatenation of its two children. The leafs are the elements of the set and the root of this tree commits to the entire set. Set membership can be proven by releasing an element's *authentication path* -- the set of hashes needed to traverse from leaf to root.

The natural way to compute a Merkle root from an array of data elements is to pass over it once, storing only the roots of subtrees, and merging them whenever possible. The intermediate state of this algorithm is a *Merkle mountain range* and it is itself another data structure for authenticating set membership. In contrast to the Merkle tree, a Merkle mountain range (MMR) is capable of accumulating new elements. The peaks of an MMR commit to the growing data set; the technical term for such a commitment is an *accumulator*.

Merkle trees and Merkle mountain ranges are dynamic in the sense that it is possible to update the tree or the mountain range as data elements change. The old data element, its authentication path, and the new data element suffice to authenticate this change. The new root or mountain peaks can be inferred from this data.

Neptune uses a Merkle Mountain Range Accumulator to represent the set of all UTXOs, both spendable and unspendable. New UTXOs are added to the accumulator. When a UTXO is spent, it is marked as such.

Transaction Processing

UTXO Structure. A UTXO is an RLWE commitment to:

- `coins` : a dictionary whose keys are `coin_type`s and whose values are `coin_state`s. For coins that determine amounts, the amount is encoded in the `coin_state`.
- `lock_script` : a script object

A `coin_type` can be derived from a `type_script` (another script object) simply by hashing it.

The arguments both to the `lock_script` and to the `type_script` are:

- the spending transaction, without witness data
- public scripts of the spending transaction
- secret witness data
- the hash of the block that generated the UTXO
- the spending transaction's timestamp

Transaction Structure. A transaction consists of the fields:

- `inputs` a list of input UTXOs
- `outputs` a list of output UTXOs
- `fee` a non-negative dyadic rational number with arbitrary precision
- `public_scripts` which should be available to anyone
- `proof` a zk-SNARK of validity
- `timestamp` an integer

The `public_scripts` field can contain any number of elements, and these elements need not *compute* anything -- they can also announce data and return `True`. The block that includes the transaction can only be valid if all public scripts return `True`. Since this validity pertains to the block and not to the transaction, we include this validity rule when describing the validity rules for blocks.

The SNARK recursion that admits coinjoin and cut-through allows increasing the timestamp. Transactions that cannot be confirmed after a certain time require a `public_scripts` element that enforces this quality.

Note that the MMR authentication paths of the input UTXOs are not part of the transaction. The reason for this omission is twof-old. First, the same transaction should still be valid if it is not confirmed and other transactions (that spend different UTXOs) are. In this event, the MMR authentication paths may need to be updated. Second, the inputs of a transaction may be outputs of another transaction that has not been confirmed yet. In this case the inputs were not yet added to the MMR and therefore don't have authentication paths. Nevertheless, merging transactions with cut-through should be a viable option.

Transaction Validity -- Base Case. A transaction is valid only if the `proof` field is a valid zk-SNARK for the following statement:

- For all input UTXOs `u`,
  - `u.lock_script(...)` accepts; and

Archived copy · cryp2nova.com

    - for all `(k,v)` in `u.coins`
        - there is a script `type_script` such that `H(type_script) = k` and `type_script(...)` accepts;

Transaction Validity -- Recusion Step. A transaction is also valid if the `proof` field is a valid zk-SNARK for the following statement:

- There are two transactions `t1` and `t2` and a UTXO set `cut_utxos` such that
    - `cut_utxos = t1.inputs ∩ ⋂ t2.outputs ∨ ∨ t2.inputs ∩ ⋂ t1.outputs`
    - `inputs` = $t1.inputs \cup t2.inputs \setminus cut_u txos$ t1.inputs ∪ t2.inputs \ cututxos
    - `outputs` = $t1.outputs \cup t2.outputs \setminus cut_u txos$ t1.outputs ∪ t2.outputs \ cututxos
    - `public_scripts` = = `t1.public_scripts ∪ ∪ t2.public_scripts`
    - `timestamp` ≥ ≥ `t1.timestamp` and `timestamp` ≥ ≥ `t2.timestamp`
    - `t1.proof` is valid, and `t2.proof` is valid

This logic captures coinjoin as well as cut-through, and even "just" increasing the transaction's timestamp.

The merge operation on transactions does not validate the `public_scripts` field. The output of the merger contains the union of `public_scripts` fields, in a random order.

Blocks

Consensus Mechanism. One of the problems with blocks that are too large or too frequent is that they take a while to propagate through the network. For a short while after one miner finds a block, the other miners waste time and effort extending the old chain -- computational effort that would be better be spent if it were securing the network.

A pipelined approach avoids propagation delays that originate from missing or yet-to-be-transmitted transactions because the validity of a block does not depend on new transactions. In Neptune, a block contains only one transaction and it includes a SNARK proving its validity. Therefore new transactions cannot induce a delay in block propagation. A receiving node does not need to be in possession of transaction data prior to receiving a new block in order to validate it and contribute to the consensus protocol. Therefore, pipelining is superfluous.

It is worthwhile to integrate awareness of the orphan rate. Blocks refer to uncles whenever they exist. This awareness serves two functions. First, the proof-of-work contributed by the uncle blocks is counted when adjusting the target difficulty. Second, a low orphan rate signals that the network operating within its capacity to process transactions. To improve transaction throughput, the block size or interval can be safely adjusted as long as the orphan rate is below a target threshold. Neptune uses adopts the former strategy: it adapts the block size dynamically but keeps the target interval constant.

Synchronizing. The long synchronization time for newcomers to the network represents a major obstacle for adoption. In Neptune, every block contains enough information for newcomers to the network to synchronize fast -- in time independent of the age or popularity of the network.

Every block comes with a number of claims which, if true, enable this fast synchronization. These claims are proved by a SNARK. Specifically:

- The predecessor block is valid. Additionally, the total amount of accumulated work is an explicit field in the block, and the SNARK establishes that it is correct. A network node who is presented with two views of history can simply decide in favor of the block where this field is larger.
- The block includes the Merkle Mountain Range Accumulator of UTXOs as well as the authentication paths of the input UTXOs of the confirmed transaction. The SNARK establishes that the MMRA was updated correctly starting from the previous block's MMRA. The MMR authentication paths in the block enable users to update the MMR authentication paths of other UTXOs.

In addition, the SNARK establishes the correctness of claims that do not pertain to fast synchronization:

- The confirmed transaction is valid.
- The coinbase output UTXO is valid.
- The output UTXOs, along with the coinbase UTXO, were correctly mixed.

Block Data Structure.

A block contains the following fields.

- `version_bits` -- used to signal membership or support
- `timestamp` -- time when the block was found
- `height` -- number of blocks between this and the genesis block, not counting the genesis block but counting the current
- `nonce` -- random bits
- `predecessor` -- hash of the previous block
- `predecessor_proof` -- SNARK that establishes that the previous block was valid
- `uncles` -- list of hashes of blocks not on the main chain
- `accumulated_pow_line` -- estimate of the number of hashes that the previous block represents
- `accumulated_pow_family` -- same as above but including orphaned blocks
- `target_difficulty` -- regulates the difficulty of finding a new block
- `max_size` -- determines how big a block can be
- `retarget_proof` -- SNARK that establishes that the target difficulty and maximum size have been computed correctly, *i.e.*, according to the parameter adjustment algorithm

Archived copy · cryp2nova.com

- `transaction` -- a single transaction that represents the merger (with coinjoin and cut-through) of all transactions being confirmed in this block
- `mixed_utxos` -- list of UTXOs whose number is one larger than all output UTXOs in `transaction`; the difference is the coinbase UTXO which contains the block reward
- `mix_proof` -- SNARK that establishes correct mixing
- `utxo_mmra` -- new Merkle Mountain Range Accumulator for UTXOs, obtained by adding the mixed UTXOs to the previous block's Merkle Mountain Range Accumulator and by marking as spent the incoming UTXOs to the confirmed transactions
- `utxo_mmra_update` -- update data for the MMRA

Block Validity.

A block is valid if

- it is the genesis block

or (all of):

- the block height is one more than that of its predecessor
- all the SNARKs are valid, including the transaction proof
- the MMRA update data is valid
- the hash of the block, cast to an integer, is less than the target difficulty
- the timestamp is no more than 15 minutes in the future
- all `public_scripts` evaluate to `True`

Block Parameter Adjustment

Producing a block requires computing SNARKs, which is a computation that takes a while to complete. In order to avoid benefiting the faster miner over the more energy-efficient one, Neptune fixes the expected block interval to a constant time large enough to produce the SNARKs. The variable parameter is the maximum block size, which can increase as long as the orphan rate is below a target threshold. The target difficulty is adjusted in order to effect the (constant) target expected block interval.

Native Currency

The Neptune native currency token type is the hash of a `type_script` that has the following logic. This logic is relative to the wildcard `$this_token_type` which refers to this native currency token type.

```
def type_script(token_type, inputs, outputs):
  input_sum = 0
  output_sum = 0

  for i in inputs:
    if token_type is in i.coins.keys():
        amount = i[$this_token_type] # dyadic rational number
        input_sum += amount

  for o in outputs:
    if token_type is in o.coins.keys():
      amount = o[$this_token_type] # dyadic rational number
      # positive output amounts
      if amount < 0:
        return False
      output_sum += amount

  # no inflation
  if output_sum > input_sum:
    return False

  return True
```

The logic makes no stipulations concerning the inflation policy. This is where the block's re-randomized UTXO set comes into play. The number of outputs of this re-randomized UTXO set is one larger than the number of inputs. The difference is the coinbase UTXO. Its validity is established by the block's correctness proof, and is determined relative to the following logic:

```
def verify_mining_reward(block, coinbase_utxo):
  epoch = floor(block.height / $halving_period)
  mining_reward = 2^(-1-epoch) * $minable_supply_limit / $halving_period + block.transaction.fee

  if mining_reward <= coinbase_utxo.coins[$native_currency_type]:
    return False

  return True
```

This logic refers to the following parameters.

Archived copy · cryp2nova.com

- `$minable_supply_limit` is the asymptotical limit of the minable token supply, arbitrarily set to 41'168'400. (Together with the 831'600 premine tokens (1.98%) this makes for a nice round number of 42'000'000.)
- `$halving_period` is the number of blocks between halvings
- `$native_currency_type` is a wildcard that refers to the native currency token type.

Dispute Resolution

Dispute resolution happens chiefly through the `public_scripts` field. In addition to computing verification logic, a public script can supply data such as hash preimages necessary to unlock a hash time lock contract.

One of the possible effects of a `lock_script` as well as `public_scripts` fields is to require the passage of a certain amount of time before an UTXO can be spent but after its generating transaction is confirmed. This time window gives the counterparty in a disputable transaction the opportunity to file their dispute, by broadcasting another transaction. The disputation transaction spends the same UTXO but avoids the passage of time criterion by activating an alternate spending policy, for instance one that involves a signature provided by the counterparty.

If the disputation transaction is broadcast before the disputed transaction is confirmed, a cut-through operation can eliminate the intermediate UTXO. This operation does not eliminate `public_scripts` or their data. The purpose of the public data is to provide the counterparty with cryptographic evidence needed to ground their dispute and to unlock the fast spending policy. In this way, the consensus mechanism is leveraged to solve the data availability problem.

If the disputation transaction is itself disputable, there will be another time window and opportunity for dispute.

Public-Facing Accounts

Neptune does not have native support for accounts. However, accounts can be simulated using the UTXO model. They are particularly relevant for public one-way access accounts; in particular, public-facing deposit accounts and public-facing withdrawal accounts.

A public-facing account is an UTXO containing a balance of any token, for instance the native currency token or even a basket of tokens. The `lock_script` constrains the ways in which the UTXO can be spent. In particular, it can only be spent to another UTXO that has the same `lock_script`. The new UTXO can have a balance delta, and the `lock_script` determines which balance deltas are allowable.

- If anyone can increase the balance, then it is a *public-facing deposit account*. The account is typically owned by user or set of users who can only spend if they provide a signature. An example of a public-facing deposit account might be the donation account of an online content producer.
- If decreasing the balance does not require a signature but instead the satisfaction of impersonal constraints, it is a *public-facing withdrawal account*. The constraints being satisfied could be winning a bet, solving a puzzle to collect the bounty, burning or marking a token to redeem the represented value, etc.

It is important to note that the new UTXO is indistinguishable from any other UTXO. In order for anyone to use it, they must be aware of the script data and randomness it commits to. The point is that this script data and randomness evolves in unpredictable ways. A naïve implementation results in a public-facing account that can be used only one, because the first user is the only one who is possession of the script data and randomness that is needed to spend the UTXO a second time.

There are several possible solutions to this *data availability problem*:

- The update data is disseminated via an off-chain protocol. The `lock_script` requires a cryptographic proof, such as a signature from the recipient, or such as a timestamped proof-of-payment to a data-availability service. This proof cannot be obtained unless the update data has been duly disseminated.
- The update data is included in one of the transaction's `public_scripts`.
- The evolution of the data is deterministic and has a variability small enough to be brute forced.

Tokenization

Neptune has native support for tokenization, as a consequence of the explicit `type_script`. A token is a tradeable quantity of units that are an alternative unit to the native currency, but indistinguishable from them in terms of operational mechanics as well as privacy aspects. The unit is derived from the `type_script` that defines the *evolutionary logic* for the token: how it is minted, how it evolves as it is traded or as a function of external events, how it is destroyed, *etc*. Two `type_script`s can define distinct tokens types despite inducing identical evolutionary logic. In this case, the `type_script` incorporates a unique identifier or random string that has no effect on the logic.

Applications

Neptune provides the base layer. A lot of functionality comes from optional plugins or apps that form overlay networks whose communication typically takes place off chain. Neptune clients provide access to an *app store*, to be curated by the client maintainer or entirely separately from the client's maintenance.

The apps themselves are written in Triton, augmented with instructions for remote procedure calls into dependee apps, into the base Neptune client, and for input and output. The use of the VM in this context has obvious motivations:

- The VM provides a level of containment for nefarious apps and app exploits.
- The apps can be executed on the VM independent of the hardware.

Archived copy · cryp2nova.com

But why not use an out-of-the-box VM for a well established language? The Triton VM offers several advantages:

- A large part of the codebase for Neptune can be recycled for apps.
- Triton programs come with a free compiler to zk-SNARK prover and verifier.
- Efforts to make Triton formally verified will carry over from Neptune to apps and vice versa.
- The easy SNARKification can be leveraged to put fair bounties on exploits, thereby guaranteeing correctness and security when formal verification is out of budget.
- The Triton VM already has optimizations for common Neptune routines, and apps can benefit from reusing these.

In the following we describe several applications (in the original sense of the word: use cases). The purpose is mainly to illustrate the mechanics and power of the Neptune base layer. In order to realize these applications, developers have to write an deploy an app on the app store.

Chess Bet

Alice and Bob want to play a game of chess, and bet on who is going to win. They both contribute one coin, and whoever ends up winning gets both of them. In case of a draw, they each get their coin back. Most importantly, Alice and Bob want to realize this chess bet without having to trust anyone else. We first describe how Alice and Bob achieve this trustless chess bet by recording every move on the blockchain; later on we will show how to move this game off-chain.

The workflow is as follows. Alice and Bob write a script (consisting of opcodes defined for the blockchain) with the following properties:

- There is a state variable that has one of five possible values: "Alice's turn", "Bob's turn", "Alice won", "Bob won", "draw".
- There is a state variable that encodes the state of the chess board.
- Alice's and Bob's public keys are hardcoded parameters.
- If it is Alice's turn, then she can use her secret key to sign a move, which updates the board state if the move is valid. Likewise for Bob. (So the script must encode the rules of chess.)
- The game state variable is updated automatically according to the move.

A commitment to this game logic will be incorporated into an Edge as the basis for the lock script. It restricts how the coins can be spent depending the game state.

- If the game is still ongoing, then the output UTXO must have the same lock script.
- If Alice won then the whole transaction must be signed under Alice's public key, and likewise for Bob.
- If the game ends in a draw, 50% of the coins can be spent to whatever outputs Alice signs; the remaining coins go to a placeholder UTXO where Bob can spend everything. The reverse order is allowed also.

Once the lock script for this program logic has been has been compiled, Alice and Bob make a transaction where they both spend one coin into the UTXO with this lock script. Then they move pieces by broadcasting transactions to that effect. When the game ends, the winner can spend their winnings.

A subtle issue arises when the state is not publicly readable on the blockchain. When Bob broadcasts a transaction and updates the commitment to the game state accordingly, how does Alice know which move Bob played? This is a specific instance of the *data availability problem*. The solution is to encode the move in the public data field of the transaction. The zk-SNARK guarantees that the move must correspond with a valid evolution of the game state.

In order to play this game off-chain, Alice and Bob set up a lightning channel whose output is spendable according to the above policy. Instead of broadcasting every transaction to the entire network and recording it on chain, they exchange transactions privately. In the event of a dispute in need of on-chain arbitration, cut-through guarantees that the dispute can be settled by broadcasting a single transaction -- albeit with multiple moves encoded into its public witness data field.

Private Multi-Token Channel Networks

Payments can take place entirely off-chain between any pair of participants in a payment channel network, provided that there is path connecting them with enough capital committed. The type of the token that is transferred can be Neptune's native currency token, or it can be any custom user-defined token. We describe here the mechanics of the token channel, and make abstraction of the token type.

In order to establish a token channel, Alice and Bob each make payment into a UTXO. When the UTXO is spent, the channel is closed and Alice and Bob's respective balances are put under their respective control. Outside of the context of multi-hop payments, the UTXO can be spent in one of three ways:

- *Mutual close*: with both Alice's and Bob's consent; in this case there is no restriction on the next transaction.
- *Unilateral close*: with only Alice's or Bob's initiave; in this case the the next transaction can only be confirmed a fixed period of time after the closure transaction -- this time is the *disputation period*. At any point in time, both Alice and Bob are in possession of the necessary information to close the channel unilaterally to the most recent agreed-upon balance.
- *Contested close*: after Alice initiated a unilateral close, Bob has the opportunity to dispute the transaction, for instance if it was closed to a deprecated balance. This disputation transaction may induce a disputation period but can only be confirmed if Alice published a unilateral channel close first.

There is a fourth way in which the UTXO can be spent, which is specific to the context of an in-progress multi-hop payment. The *hash time lock contract (HTLC)* allows the funding UTXO to be spent in a way that confirms the balance (assuming the payment goes through) conditional upon the provision of a hash preimage. The node who is on the receiving side of a channel in a payment path can publish this transaction and lock in the updated balance if he is knowledgeable of the hash preimage. The sender in that channel, if he wants to avoid closing the channel, will agree to update the channel off-chain and

Archived copy · cryp2nova.com

sign the necessary data to that effect. In exchange, he receives the hash preimage, which he can use to exact the off-chain balance update one hop up the line. In this way, the transmission of the preimage enforces the correct balance updates across the entire payment path.

One way to achieve this functionality in Neptune is through the `public_scripts` field of transactions. One public script contains logic for verifying a hash digest against that of an unknown preimage. The logic is satisfied only if another `public_scripts` element supplies the preimage.

An alternative way to achieve this functionality is by putting the logic into the `lock_script`. The resulting proof will then be valid only if the containing transaction has a `public_scripts` element that supplies the preimage.

Multiple Tokens.

The same construction applies to custom user-defined tokens and not just the native Neptune currency. Any token can be transferred over a channel network that supports it. Such a *multi-token channel network* supports lightning fast token trades with near-instant settlement. Every node becomes a high-frequency trading marketplace.

Flash Arbitrage.

An arbitrage opportunity occurs when there is a cycle of trades that results in a net positive position without risk to the trader. In order to take advantage of it, the traditional trader must run through the cycle of trades. In Ethereum, a flash loan is when an arbitrage trader takes out a loan, runs through the cycle of trades, and repays the loan, all in a single block. In a multi-token channel network, the arbitrage trader does not need to take out the loan to begin with; he merely coordinates the cycle of balance updates and activates them all in one atomic off-chain trade.

Non-Fungible Tokens

A non-fungible token (NFT) is a transferable digital record whose amount is limited to one. It typically represents something, such as the legal rights to a digital piece of art. In terms of blockchain logic, this representation is vacuous. However, an overlay legal system may recognize and enforce the resulting legal claims.

An NFT can only be unique with respect to a central anchor point common to all parties. The anchor points themselves can be generated arbitrarily by anyone, but only sufficiently popular anchor points can serve the purpose of anchoring NFTs.

In privacy-preserving cryptocurrency, NFTs pose a unique challenge. If it is difficult to distinguish NFT-minting transactions from ordinary ones, what's to stop a malicious actor from minting an NFT for the same asset twice?

On Neptune, a natural choice is to use a unique token type for every concrete NFT. The NFTs are anchored to a central point which is itself a token of a special type. The anchor token can be spent by anyone as long as the spending conditions remain unchanged. Its state represents an integer that is incremented by one every time it is spent.

The NFT itself inherits the state of the NFT anchor, but adopts this state as a unique identifier that cannot be changed. The `type_script` of the NFT binds this identifier to the asset's unique bit string representative. This linkage is confirmed by the producer's signature, which is another field of the NFT's state. Ownership is changed via ordinary `lock_script`s, making NFTs indistinguishable from other tokens on the network except by those individuals that know the secret keys that unlock them.

The NFT's `type_script` further constrains the events that can generate them. Either the NFT exists as both an input and an output to the transaction, or else it is generated in the transaction and in this case the NFT generator exists as both an input and an output to the transaction.

Tokenized Derivative Positions

In finance, a *derivative* is a contract whose execution is contingent on market events such as the price of a particular asset reaching or crossing a threshold level. Mechanically, it is similar from a gamble; the big difference is that the success criterion is not pure luck.

A derivative contract needs several components:

- Parties to the contract. The parties' positions are complementary in the sense that one participant stands to win what the other stands to lose. Furthermore, the parties need to be bound by the contract: they should be unable of escaping their obligations in case they should lose.
- Contract logic, which specifies the conditions under which the contract becomes exercisable.
- Market data, typically provided through an oracle. We make abstraction of how the oracle operates.

These components have a straightforward translation into the logic of Neptune's smart contract ecosystem. The parties lock funds into an UTXO whose `lock_script` stipulates that it can only be spent by one of two tokens, and only after the contract is exercisable. The tokens themselves are minted when the funds are locked, and the winning token is burned when the funds are withdrawn. Anyone can mint pairs of tokens.

The `lock_script` furthermore establishes an authentication mechanism that determines which oracle inputs can be used to claim victory. Several options come to mind:

- The oracles' public keys are hardcoded into the `lock_script`; their data is only valid if it comes with a matching signature.
- An on-chain order book or market maker sets the price, and this price feed is authenticated by the derivative contract.
- Oracle data is supplied by tokens involved in the transaction, which may or may not be burn-after-using.

Archived copy · cryp2nova.com

Cryptosecurities

In finance, a *security* is a contract that confers unto the holder a right that indirectly translates into a monetary value. The keyword is *indirectly* -- since this translation depends on the performance of particular clauses of the contract by a party to the contract. For instance, the shareholder is entitled to a proportion of the dividend in case of dividend payment; of the assets in case of liquidation. The bond holder is entitled to principal plus interest, but only in the future and only if the counterparty is otherwise solvent. The mortgage lender is entitled to regular installments over a certain period, and forsakes his claim to the collateral asset if all these installments were made on time.

Cryptocurrencies offer two improvements. First, they disintermediate -- since the contracts are executed mechanically, there is no need for a middleman to coordinate or arbitrate. Second, they enable activation of contract clauses while leaking zero knowledge. The observer and perhaps even the account owner learns nothing about which article is being employed, only that the activity is consistent with the contract.

This explanation benefits from a concrete example. Suppose Alice wishes to take out a mortgage on her house. The ownership of the house has been tokenized -- that is to say, there is an NFT that represents the house and the local law enforcement recognizes and enforces this property claim.

To activate the loan, Alice and the bank simultaneously publish two transactions. The first moves the borrowed money into Alice's control. The second creates the deposit account where the monthly payments go. Specifically, the output UTXO of this second transaction contains the house NFT and some amount (initially zero) of native currency. This UTXO can only be spent to another similar UTXO, with any of the following differences.

- Only the bank can reduce the balance of native currency and it can do so without restriction.
- Anyone can increase the native currency balance but only if the resulting UTXO is signed by the bank. (This signature guarantees data availability -- in particular, it guarantees that the bank is aware of the new state.)
- The house NFT can be spent by Alice only if she provides proof that all her monthly payments have been made on time.
- The house NFT can be spent by the bank only if Alice missed one of her monthly payments.

This example refers to a bank as though it is a single centralized party. In fact, there can be multiple separate lenders that are not required to trust each other. To achieve this, the lenders mint security tokens in proportion to their share of the borrowed amount. Money can be spent from the deposit account only in exchange for burning an equal amount of security tokens. The tokens can be traded privately since they are indistinguishable from other assets.

Anonymous Fundraising and Investment

Stocks are a special type of security, and so the same general construction as above applies. A startup can establish a company deposit account UTXO, which can only be spent to another UTXO if it has the same spending criteria but possibly different native currency balances. Specifically:

- Investors can increase the native currency balance in exchange for shares, which is a token type tailored to the company or foundation.
- The company or foundation administrators can decrease the native currency balance without restriction unless liquidation is activated.
- In the event of liquidation, the shareholders can burn shares in exchange for withdrawing a proportional amount from the native currency balance.

Companies can issue dividends by transferring them into a dividend pot, which is an UTXO that can only be spent by shareholders. Specifically, in order to make a withdrawal from this account, a shareholder must do two things.

1. He must prove that he is entitled to the withdrawal. This step boils down to using the shares to generate a proof of entitlement that unlocks the spending criterion.
2. He must mark his share as having received the dividend in question. This mark prevents the same token from being used twice. The mark itself is stored in the token's state, which is updated in accordance with the token's `type_script`.

Neptune allows investors to invest and collect dividends anonymously.

© 2026 Neptune Cash Developers. All rights reserved. Anonymous, cookie-free analytics are used to improve content.

Archived copy · cryp2nova.com