Ergo: A Resilient Platform For Contractual Money

Ergo Developers, https://ergoplatform.org

May 14, 2019
v1.0

Abstract

We present Ergo, a new flexible blockchain protocol. Ergo is designed for developing decentralized applications with the main focus of providing an efficient, secure and easy way to implement financial contracts.

To achieve this goal, Ergo includes various technical and economic improvements to existing blockchain solutions. Every coin in Ergo is protected by a program in ErgoScript, which is a powerful and protocol-friendly scripting language based on Σ-protocols. Using ErgoScript, we can encode the conditions under which coins may be used: who can spend them, when, under what external conditions, to whom, and so on.

Extended support for light nodes makes Ergo friendly for end users because it allows running contracts on untrusted commodity hardware. To be usable in the long-term, Ergo follows a survivability approach – it uses widely-researched solutions that don't result in security issues in the future, while also preventing performance degradation over time with a new economic model. Finally, Ergo has a self-amendable protocol that allows it to absorb new ideas and improve itself in the future.

1 Introduction

Beginning more than ten years ago with Bitcoin [1], blockchain technology has so far proved to be a secure way of maintaining a public transaction ledger and disintermediating trusted third parties such as traditional financial institutions to some degree. Even after achieving a market capitalization over $300bn in 2017 [2], no severe attacks were performed on the Bitcoin network despite the high potential yield. This resilience of cryptocurrencies and the financial empowerment and self-sovereignty they promise to bring is achieved by a combination of modern cryptographic algorithms and decentralized architecture.

However, this resilience comes at a cost and has not yet been proven for existing systems in the long-run at economy-wide scale. To use a blockchain without any trust, its participants should check each other by downloading and processing all the transactions in the network, utilizing network resources.

Archived copy · cryp2nova.com

Besides network utilization, transaction processing also utilizes computational resources, especially if the transactional language is sufficiently flexible. Finally, blockchain participants have to keep a significant amount of data in their local storages and the storage requirements are growing fast. Of this, certain data must be maintained in memory. Thus, transaction processing utilizes various resources of hundreds of thousands of computers all over the world and consumption of these resources is paid for by regular users in the form of transaction fees [3]. Despite the generous block reward subsidy in some existing systems, their fees can still be very high at times [4]. Due to this, even after being around for more than ten years, blockchain technology is still primarily being used in financial applications, where the advantage of high security outweighs the disadvantage of high transaction costs.

Besides the vanilla currency example, the other use of blockchains is to build decentralized applications. Such applications utilize the ability of the underlying platform to write smart contracts [5] implementing their logic by means of a blockchain-specific programming language. One way to classify blockchains in terms of their ability to write smart contracts is based on if they are *UTXO-based* (e.g., Bitcoin) or *account-based* (e.g., Ethereum) [6]. Account-based cryptocurrencies, such as Ethereum, introduce special contract accounts controlled by code, that may be invoked by incoming transactions. Although this approach allows performing arbitrary computation, the implementation of complex spending conditions can lead to bugs such as the one in an Ethereum's "simple" multi-signature contract that caused a loss of $150 million in 2017 [7]. In UTXO-based cryptocurrencies, every coin has a script associated with it, and to spend that coin, one must satisfy the conditions given in the script. Implementing such protecting conditions is much easier with the UTXO model but doing arbitrary Turing-complete computation is quite complex [8]. However, most financial contracts do not require Turing-completeness [9]. Ergo is based on the UTXO model and provides a convenient way to implement financial applications covering an overwhelming majority of public blockchain use-cases.

While the contractual component is important for building decentralized applications, it is also essential that the blockchain survives in the long-term. Application-oriented blockchain platforms have existed only for a few years and the whole area is quite young. Since such platforms have already encountered problems with performance degradation over time [10, 11], their long-term survivability is questionable. Even older UTXO-based money-oriented blockchains have not been proven to be fully resilient in the long-run under changing conditions because we only have about 10 years of blockchain history up to this point. Solutions for long-term survivability include concepts such as light nodes with minimal storage requirements [12], storage-rent fee component to prevent bloating of full-nodes [3], and self-amendable protocols that can adapt to the changing environment and improve themselves without trusted parties [13]. What is needed is a combination of various scientific ideas together to fix these problems, while also providing a way for further improvements without any breaking changes and this is exactly what Ergo seeks to accomplish.

Archived copy · cryp2nova.com

2 Ergo Vision

The Ergo protocol is very flexible and may be changed in the future by the community. In this section, we define the main principles that should be followed in Ergo which might be referred to as "Ergo's Social Contract". In case of intentional violation of any of these principles, the resulting protocol should not be called Ergo.

- *Decentralization First.* Ergo should be as decentralized as possible: any parties (social leaders, software developers, hardware manufacturers, miners, funds and so on) whose absence or malicious behavior may affect the security of the network should be avoided. If any of these parties do appear during Ergo's lifetime, the community should consider ways to decrease their impact level.
- *Created for Regular People.* Ergo is a platform for ordinary people, and their interests should not be infringed upon in favor of big parties. In particular, this means that centralization of mining should be prevented and regular people should be able to participate in the protocol by running a full node and mining blocks (albeit with a small probability).
- *Platform for Contractual Money.* Ergo is the base layer to applications that will be built on top of it. It is suitable for several applications but its main focus is to provide an efficient, secure and easy way to implement financial contracts.
- *Long-term Focus.* All aspects of Ergo development should be focused on a long term perspective. At any point of time, Ergo should be able to survive for centuries without expected hard forks, software or hardware improvements or some other unpredictable changes. Since Ergo is designed as a platform, applications built on top of Ergo should also be able to survive in the long term. This resiliency and long term survivability may also enable Ergo to be a good store of value.
- *Permissionless and Open.* Ergo protocol does not restrict or limit any categories of usage. It should allow anyone to join the network and participate in the protocol without any preliminary actions. Unlike the traditional financial system, no bailouts, blacklists or other forms of discrimination should be possible on the core level of Ergo protocol. On the other hand application developers are free to implement any logic they want, taking responsibility for the ethics and legality of their application.

3 Autolykos Consensus Protocol

The core component of any blockchain system is its consensus protocol and Ergo utilizes a self-developed unique Proof of Work (PoW) consensus protocol called *Autolykos*, which is described below. Despite extensive research on possible

Archived copy · cryp2nova.com

alternatives, the original PoW protocol with the longest chain rule is still in demand due to its simplicity, high-security guarantees, and friendliness to light clients. However, a decade of extensive testing has revealed several problems with the original one-CPU-one-vote idea.

The first known problem of a PoW system is the development of specialized hardware (ASICs), which allows a small group of ASIC-equipped miners to solve PoW puzzles orders of magnitude faster and more efficiently than everyone else. This problem can be solved with the help of memory-hard PoW schemes that reduce the disparity between ASICs and commodity hardware. The most promising approach here is to use asymmetric memory-hard PoW schemes that require significantly less memory to verify a solution than to find it [14, 15].

The second known threat to a PoW network decentralization is that even big miners tend to unite in mining pools, leading to a situation when just a few pool operators (5 in Bitcoin, 2 in Ethereum at the time of writing) control more than 51% of computational power. Although the problem has already been discussed in the community, no practical solutions have been implemented before Ergo.

Ergo's PoW protocol, Autolykos [16], is the first consensus protocol that is both memory-hard and pool-resistant. Autolykos is based on the *one list k-sum problem*: a miner has to find $k = 32$ elements from a pre-defined list R of size $N = 2^{26}$ (which has a size of 2 Gb), such that $\sum_{j \in J} r_j - sk = d$ is in the interval $\{-b, \ldots, 0, \ldots, b \mod q\}$. Elements of list R are obtained as a result of one-way computation from index i, two miner public keys pk, w and hash of block header m as $r_i = H(i||M||pk||m||w)$, where H is a hash function which returns the values in $\mathbb{Z}/q\mathbb{Z}$ and M is a static big message that is used to make hash calculation slower. Also, a set of element indexes J is to be obtained by one-way pseudo-random function $genIndexes$, that prevents possible solutions search optimizations.

Thus, we assume that the only option for a miner is to use the simple brute-force method given in Algorithm 1 to create a valid block.

Algorithm 1 Block mining

Input: upcoming block header hash m, key pair $pk = g^{sk}$
Generate randomly a new key pair $w = g^x$
Calculate $r_{i \in [0,N)} = H(i||M||pk||m||w)$
while *true* **do**
    $nonce \leftarrow$ rand
    $J := genIndexes(m||nonce)$
    $d := \sum_{j \in J} r_j \cdot x - sk \mod q$
    if $d < b$ **then**
        return $(m, pk, w, nonce, d)$
    end if
end while

Note that although the mining process utilizes private keys, the solution itself only contains public keys. Solution verification is done by Algorithm 2.

Archived copy · cryp2nova.com

Algorithm 2 Solution verification

1: **Input**: $m, pk, w, nonce, d$
2: require $d < b$
3: require $pk, w \in \mathbb{G}$ and $pk, w \neq e$
4: $J := genIndexes(m||nonce)$
5: $f := \sum_{j \in J} H(j||M||pk||m||w)$
6: require $w^f = g^d \cdot pk$

This approach prevents mining pool formation because the secret key sk is needed for mining: once any pool miner finds a correct solution, he can use this secret to steal the block reward. On the other hand, it is secure to reveal a single solution, as it only contains public keys and reveals a single linear relation between the 2 secrets sk, w.

Memory-hardness follows from the fact that Algorithm 1 requires keeping the whole list R for the main loop execution. Every list element takes 32 bytes, so the whole list of N elements takes $N \cdot 32 = 2Gb$ of memory for $N = 2^{26}$. A miner can try to reduce memory requirements by calculating these elements "on the fly" without keeping them in memory, however, he'll need to calculate the same hash H multiple times (about 10^4 times for modern GPUs), thereby reducing efficiency and profit.

Calculating the list R is also quite a heavy computational task: our initial implementation [17] consumes 25 seconds on Nvidia GTX 1070 to fill all the 2^{26} elements of the list. This part, however, may be optimized if a miner also stores a list of unfinalized hashes $u_{i \in [0,N)} = H(i||M||pk)$ in memory, consuming 5 more Gigabytes of it. In such a case, work to calculate unfinalized hashes should be done only once during mining initialization while finalizing them and filling the list R for the new header only consumes a few milliseconds (about 50 ms on Nvidia GTX 1070).

The target parameter b is built-in into the puzzle itself and is adjusted to the current network hash rate via a difficulty adjustment algorithm [18] to keep time interval between block close to 2 minutes. This algorithm tries to predict the hash rate of an upcoming 1024 blocks long epoch based on data from the previous 8 epochs via the well-known *linear least squares method*. This makes the predictions better than that of the usual difficulty adjustment algorithm and also makes "coin-hopping" attacks less profitable.

4 Ergo State

To check a new transaction, a cryptocurrency client does not use the ledger with all the transactions that happened before this one. Instead, it uses a ledger state snapshot from its history. In the Bitcoin Core reference implementation, this snapshot is the active one-time coins (i.e., UTXOs), and a transaction destroys some coins and also creates new ones. In Ethereum, this snapshot is of long-lived accounts and a transaction modifies monetary balance and internal storage

Archived copy · cryp2nova.com

of some accounts. Also, in Ethereum (unlike Bitcoin), the representation of the snapshot is fixed within the protocol because an authenticating digest of the snapshot is written into the block header.

Ergo follows Bitcoin's UTXO design and represents the snapshots using one-time coins. The difference from Bitcoin is that in addition to monetary value and protecting script, an Ergo one-time coin, called a *box*, also contains user-defined data. Similar to Ethereum, an Ergo block also stores an authenticating digest, called the *stateRoot*, of the global state after applying the block.

An Ergo box is made of registers (and nothing but registers). Such a box can have 10 registers labeled $R_0, R_1, \ldots, R_9$, of which the first four are filled with mandatory values and the rest may contain arbitrary data or be empty.

- R_0 *(monetary value).* Amount of Erg locked in this box.
- R_1 *(guard script).* Serialized script protecting this box.
- R_2 *(tokens).* A box can carry multiple tokens. This register contains an array of $(token_identifier \rightarrow amount)$ pairs locked in this box.
- R_3 *(transaction info).* Contains (1) declared creation height (should be no more than actual height of a block which contains the transaction), (2) a unique identifier of the transaction that created this box, and (3) the index of this box in that transaction's output boxes.
- $R_4 - R_9$ *(additional data).* Contains arbitrary user-defined data.

One-time immutable objects (as in Bitcoin's UTXO model) have some advantages over Ethereum's long-lived mutable accounts. Firstly, it gives easier and safer protection from replay or reordering attacks. Secondly, it is easier to process transactions in parallel because they don't modify state of the objects they access. Also, a transaction either modifies the system state exactly as intended, or does not change it at all (with no possible side-effects resulting from 'out-of-gas' exceptions, reentrancy issues, and so on). Finally, it seems easier to build fully stateless clients using one-time coins [19] (although research in this area is still in the initial stage).

One major criticism of one-time coins is that this model does not seem suitable for non-trivial decentralized applications. However, Ergo has overcome the problems and shown this assertion to be false by demonstrating many non-trivial prototype applications built on top of it (see Section 7).

The Ergo protocol fixes the ledger snapshot representation in the form of boxes not destroyed by previous transactions. In detail, a miner should maintain a Merkle-tree like authenticated data structure built on top of the UTXO set and must include a short digest (just 33 bytes) of this structure in each block header. This digest must be calculated *after* applying the block. This authenticated data structure is built on top of an AVL+ tree [12], which like a regular hash tree, allows generating proofs of existence or non-existence of particular elements in the tree. Thus, users maintaining the full tree are capable of generating proofs that their boxes are unspent and a small 33 bytes digest is sufficient for verifying

Archived copy · cryp2nova.com

these proofs. However, unlike regular hash trees, an AVL+ tree also allows generation of proofs of tree modifications that allow verifiers to compute the new tree digest. Ergo miners are required to generate proofs of block modifications and a hash of this proof is included into the block header along with the digest of the resulting state. Hence, light nodes that only maintain a small digest of the current state are able to verify a full block – they can check that all spent boxes were removed from the state, all created boxes were added to it and no more changes were made.

AVL+ trees allow building efficient authenticated dictionaries that reduce the proof size and speed up verification by 1.4-2.5 times in comparison to prior solutions, making them better suited for the cryptocurrency applications. For instance, our proofs are about 3 times smaller than proofs of a Merkle Patricia trie used in Ethereum for the same purpose (see Figure 1).

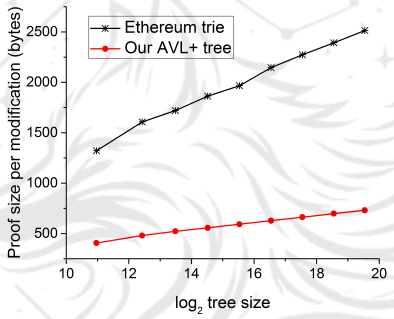


Figure 1: Proof size comparison with a Merkle patricia trie

Finally, proofs for multiple transactions in a single block are compressed together, reducing their total length by approximately an additional factor of 2:

CRYP2NOVA

Archived copy · cryp2nova.com

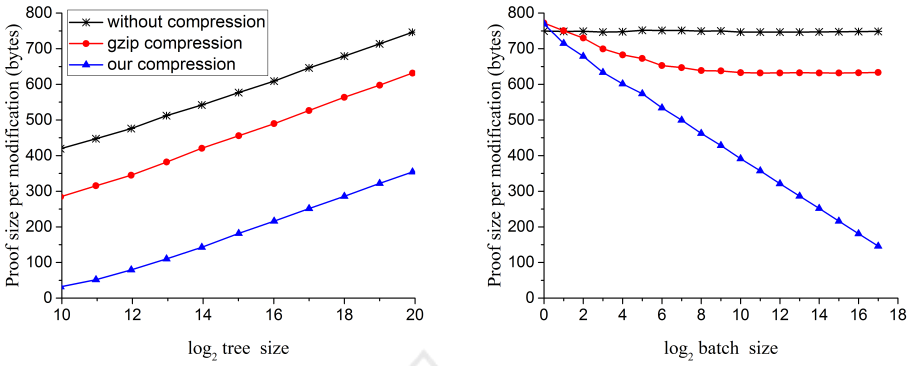


Figure 2: Left: proof size per modification for 2000 transactions as a function of starting tree size n. Right: proof size per modification for a tree with $n = 1000000$ keys as a function of batch size B. In both cases, half of the modifications were inserts of new (key, value) pairs and the other half were change of values for existing keys.

Thus, Ergo state provides an efficient and secure way to prove existence or non-existence of certain elements in it, as well as proofs of tree modifications. These tree operations are supported by the Ergo smart contract language, thereby providing the ability to implement sophisticated contracts like those discussed in Section 7.

5 Resiliency and Survivability

Being a platform for contractual money, Ergo should also support long-term contracts for a period of at least an average person's lifetime. However, even young existing smart contract platforms are experiencing issues with performance degradation and adaptability to external conditions. This leads to a situation where the cryptocurrency depends on a small group of developers to provide a fixing hard-fork, or the cryptocurrency won't survive. For example, the Ethereum network was started with a Proof-of-Work consensus algorithm with a plan to switch to Proof-of-Stake in the near future. However, delays in the Proof-of-Stake development have led to several fixing hard-forks [20] and the community is still forced to rely on core developers promising to implement the next hard-fork.

The first common survivability issue is that in pursuit of popularity, developers tend to implement ad-hoc solutions without proper preliminary research and testing. Such solutions inevitably lead to bugs, which then lead to hasty bug fixes, then to fixes of those bug fixes, and so on, making the network unreliable and even less secure. A notable example is the IOTA cryptocurrency, which implemented various scaling solutions, including its own hash function and DAG structure, that allowed it to achieve high popularity and market capitalization. However, a detailed analysis of these solutions revealed multiple serious prob-

Archived copy · cryp2nova.com

lems, including practical attacks that enabled token theft [21, 22]. A subsequent hard-fork [23] then fixed these problems by switching to the well-known SHA3 hash function, thereby confirming the uselessness of such kind of innovations. Ergo's approach here is to use stable well-tested solutions, even if they lead to slower short-term innovations. Most of the solutions used in Ergo are formalized in papers presented at peer-reviewed conferences [12, 18, 3, 8, 24, 25] and have been widely discussed in the community.

A second problem that decentralization (and thus survivability) faces is the lack of secure trustless light clients. Ergo tries to fix this problem of blockchain technology without creating new ones. Since Ergo is a PoW blockchain, it easily allows extraction of a small header from the block content. This header alone permits validation of the work done in the block and a headers-only chain is enough for best chain selection and synchronization with the network. A headers-only chain, although much smaller than the full blockchain, still grows linearly with time. Recent research on light clients provides a way for light clients to synchronize with the network by downloading an even smaller amount of data, thereby unlocking the ability to join the network using untrusted low-end hardware such as mobile phones [26, 27]. Ergo uses an authenticated state 4 and for transactions included in a block, a client may download a proof of their correctness. Thus, regardless of the blockchain size, a regular user with a mobile phone can join the network and start using Ergo with the same security guarantees as a full node.

Readers may notice a third potential problem in that although support for light clients solves the problem for Ergo users, it does not solve the problem for Ergo miners, who still need to keep the whole state for efficient transaction validation. In existing blockchain systems, users can put arbitrary data into this state. This data, which lasts forever, creates a lot of dust and its size increases endlessly over time [28]. A large state size leads to serious security issues because when the state does not fit in random-access memory, an adversary can generate transactions whose validation become very slow due to required random access to the miner's storage. This can lead to DoS attacks such as the one on Ethereum in 2016 [29]. Moreover, the community's fear of such attacks along with the problem of "state bloat" without any sort of compensation to miners or users holding the state may have prevented scaling solutions that otherwise could have been implemented (such as larger block sizes, for example). To prevent this, Ergo has a storage rent component: if an output remains in the state for 4 years without being consumed, a miner may charge a small fee for every byte kept in the state.

This idea, which is similar to regular cloud storage services, was only proposed quite recently for cryptocurrencies [30] and has several important consequences. Firstly, it ensures that Ergo mining will always be stable, unlike Bitcoin and other PoW currencies, where mining may become unstable after emission is done [31]. Secondly, growth of the state's size becomes controllable and predictable, thereby helping Ergo miners to manage their hardware requirements. Thirdly, by collecting storage fees from outdated boxes, miners can return coins to circulation, and thus, prevent the steady decrease of circulating

Archived copy · cryp2nova.com

supply due to lost keys [32]. All these effects should support Ergo's long-term survivability, both technically and economically.

A fourth vital challenge to survivability is that of changes in the external environment and demands placed on the protocol. A protocol should adapt to the ever changing hardware infrastructure, new ideas to improve security or scalability that emerge over time, the evolution of use-cases, and so on. If all the rules are fixed without any ability to change them in a decentralized manner, even a simple constant change can lead to heated debates and community splits. For instance, discussion of the block-size limit in Bitcoin led to its splitting into several independent coins. In contrast, Ergo protocol is self-amendable and is able to adapt to the changing environment. In Ergo, parameters like block size can be changed on-the-fly via voting of miners. At the beginning of each 1024-block voting epoch, a miner proposes changes of up to 2 parameters (such as an increase of block size and a decrease of storage fee factor). During the rest of the epoch, miners vote to approve or reject the changes. If a majority of votes within the epoch support the change, the new values are written into the extension section of the first block of the next epoch, and the network starts using the updated values for block mining and validation.

To absorb more fundamental changes, Ergo follows the approach of *soft-forkability* that allows changing the protocol significantly while keeping old nodes operational. At the beginning of an epoch, a miner can also propose to vote for a more fundamental change (e.g., adding a new instruction to ErgoScript) describing affected validation rules. Voting for such breaking changes continues for 32,768 blocks and requires at least 90% of "Yes" votes to be accepted. Once being accepted, a 32,768-blocks long activation period starts to give outdated nodes time to update their software version. If a node software is still not updated after the activation period, then it skips the specified checks but continues to validate all the known rules. List of previous soft-fork changes is recorded into the extension to allow light nodes of any software version to join the network and catch up to the current validation rules. A combination of soft-forkability with the voting protocol allows changing of almost all the parameters of the network except the PoW rules that are responsible for the voting itself.

6 Ergo's Native Token

Ergo platform has its native token, which is called Erg and is divisible to up to 10^9 smallest units, nanoErgs (a nanoErg is one billionth of an Erg). Ergs are important for Ergo platform stability and security by several reasons discussed below.

During the initial phase of Ergo's life, miners will receive the reward in Ergs according to a predefined and hard-coded token emission schedule (see 6.1 for more details). These coins will incentivize miners to participate in the Ergo network, securing it from hashrate-based attacks like the known 51% attack [33].

Erg emission will be finished within just eight years, and after that miners will only receive Ergs from fees. Although, adjustable over time through miner

Archived copy · cryp2nova.com

on-chain voting, Ergo block size and maximum block computational cost at any given point in time will be limited, and thus miners are enforced to choose only a subset of transactions from mempool during times of high load. Fees will help miners to sort the transactions, preventing spam attacks while allowing miners to include transactions from honest users in blocks.

Besides network and computation resources, a transaction utilizes storage by increasing state size. In existing cryptocurrencies, an element of the state, being a UTXO in UTXO-based blockchains, called a box in Ergo, once created lives possibly forever without any compensation to miners and some users who must keep this state in high-cost random-access memory. This leads to a misalignment of incentives and continuously increasing state size. In contrast, Ergo has a storage rent component that periodically charges users Erg for every byte included in the state. This storage rent is making the system more stable by limiting state size or insuring proper compensation for larger state size, returning lost coins into circulation and providing an additional stable and predictable reward to miners.

Thus, being a platform for contractual money, Ergo is suitable to build applications and monetary systems on top of it. However, participating in such systems would require using the Ergo native token, Erg, as well in order to pay storage rent and transaction fees which will provide miners strong ongoing incentives to secure the network with adequate hash power. Users, for their part, will be highly incentivized to purchase, use and save Ergs if they find applications for Ergo to be of high value.

6.1 Emission

All Erg tokens that will ever circulate in the system are presented in the initial state, which consists of 3 boxes:

- *No Premine Proof.* This box contains exactly one Erg and is protected by a script that prevents it from being spent by anyone. Thus, it is a long-lived box that will stay in the system until the storage-rent component destroys it. Its main purpose is to prove that Ergo mining was not started privately by anyone before the declared launch date. To achieve this, additional registers of this box contain the latest headlines from the media (The Guardian, Vedomosti, Xinhua), as well as the latest block identifiers from Bitcoin and Ethereum. Thus, Ergo mining could not have started before certain events in the real world and the cryptocurrency space.

- *Treasury.* This box contains 4,330,791.5 Ergs that will be used to fund Ergo development. Its protecting script [34] consists of two parts.

  First, it ensures that only a predefined portion of the box value is unlocked. During blocks 1-525,599 (2 years) 7.5 Ergs will be released every block. Then during blocks 525,600-590,399 (3 months) 4.5 Ergs will be released every block. Finally, during blocks 590,400-655,199 (3 months) 1.5 Ergs will be released every block. This rule ensures the presence of funds for

Archived copy · cryp2nova.com

Ergo development for 2.5 years and, at any moment of time, rewards do not exceed 10% of the total number of coins in circulation.

Second, it has custom protection from unexpected spending. Initially, it requires the spending transaction to be signed by at least 2 of 3 secret keys that are under control of the initial team members. When they spend the box, they are free to change this part of the script as they wish, for example by adding new members to protect foundation funds.

During the first year, these funds will be used to cover the pre-issued EFYT token. After that, they will be distributed in a decentralized manner via a community voting system that is under development.

- *Miners Reward.* This box contains 93,409,132 Ergs that will be collected by block miners as a reward for their work. Its protecting script [35] requires the spending transaction to have exactly two outputs with the following properties:
  - The first output should be protected by the same script and the number of Ergs in it should equal to the remaining miners' reward. During blocks 1 - 655,199, a miner will be able to collect 67.5 Ergs from this box. During blocks 655,200 - 719,999 (3 months), a miner will be able to collect 66 Ergs, and after that, the block reward will be reduced by 3 Ergs every 64,800 blocks (3 months) until it reaches zero at block 2,080,800.
  - The second output should contain the remaining coins and should be protected by the following condition: it can be spent by a miner that solved the block's PoW puzzle and not earlier than 720 blocks after the current block.

All of these rules result in the following curve denoting the number of coins in circulation with time:

CRYP2NOVA

Archived copy · cryp2nova.com

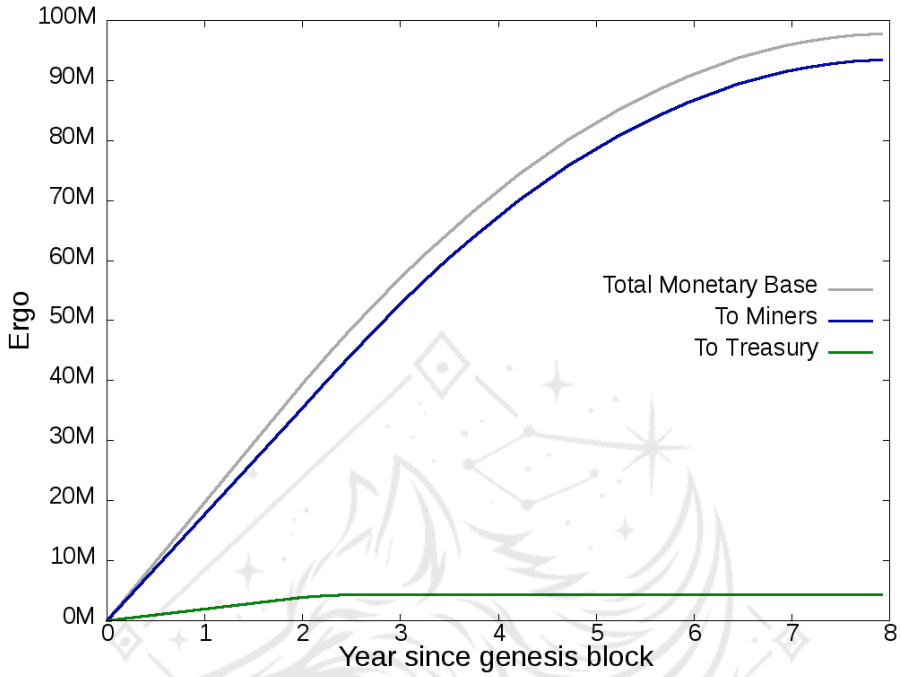


Figure 3: Ergo emission curve

7 Contractual Money

In our opinion, the overwhelming majority of use-cases for public blockchains (even those that claim to provide a general-purpose decentralized world computer) are for financial applications, which do not require Turing-completeness. For instance, if an oracle writes down non-financial data into the blockchain (such as temperature), this data is usually used further in a financial contract. Another trivial observation we make is that many applications use digital tokens with mechanics different from the native token.

For an application developer, the Ergo Platform offers custom tokens (which are first-class citizens) and a domain-specific language for writing box protecting conditions in order to implement flexible and secure financial applications. Ergo applications are defined in terms of protecting scripts built into boxes, which may also contain data involved in the execution. We use the term *contractual money* to define Ergs (and secondary tokens) whose usage is bounded by a contract. This applies to all tokens on the platform in existence because any box with its contents (Ergs, tokens, data) is bounded by a contract.

However, we can distinguish between two types of contractual Ergs. The first, called *free Ergs*, are the ones that could change their contracts easily and

Archived copy · cryp2nova.com

have no restrictions on the outputs or the other inputs of a spending transaction. The second type is *bounded Ergs*, whose contracts require the spending transaction to have input and output boxes with specific properties.

For example, if a box A is protected by just a public key (so providing a signature against a spending transaction is enough in order to destroy the box), the public key owner can spend A and transfer the Ergs to any arbitrary output box. Thus, the Ergs within A are free. In contrast, imagine a box B protected by a combination of a public key and a condition that demands the spending transaction to create an output box with the same amount of Ergs as in B and whose guarding script has the hash `rBMUEMuPQUx3GzgFZSsHmLMBouLabNZ4HcERm4N` (in Base58 encoding). In this case, the Ergs in B are bounded Ergs.

Similarly, we can define free and bounded tokens. An Ergo contract can have several hybrids such as bounded Ergs and free tokens or both bounded under one public key and free under another.

7.1 Preliminaries For Ergo Contracts

While in Bitcoin, a transaction output is protected by a program in a stack-based language named *Script*, in Ergo a box is protected by a logic formula which combines predicates over a context with cryptographic statements provable via zero-knowledge protocols using AND, OR, and k-out-of-n connectives. The formula is represented as a typed direct acyclic graph, whose serialized form is written in a box. To destroy a box, a spending transaction needs to provide arguments (which include zero-knowledge proofs) satisfying the formula.

However, in most cases, a developer is unlikely to develop contracts in terms of graphs. Instead, he would like to use a high-level language such as ErgoScript, which we provide with the reference client.

Writing scripts in ErgoScript is easy. As an example, for a one-out-of-two signature, the protecting script would be $pk_1 \| pk_2$, which means "prove knowledge of a secret key corresponding to the public key pk_1 or knowledge of a secret key corresponding to public key pk_2". We have two separate documents for help in developing contracts with ErgoScript: the "ErgoScript Tutorial" [36] and the "Advanced ErgoScript Tutorial" [37]. Thus, we do not get into the details of developing contracts with ErgoScript. Rather, we provide a couple of motivating examples in the following sections.

Two more features of Ergo shaping contracting possibilities are:

- *Data Inputs:* To be used in a transaction, a box need not be destroyed but can instead be read-only. In the latter case, we refer to the box as being part of the *data input* of the transaction. Thus, a transaction gets two box sets as its arguments, the inputs and data inputs, and produces one box set named *outputs*. Data inputs are useful for oracle applications and interacting contracts.

- *Custom Tokens:* A transaction can carry many tokens as long as the estimated complexity for processing them does not exceed a limit, a parameter

Archived copy · cryp2nova.com

that is set by miner voting. A transaction can also issue a single token with a unique identifier which is equal to the identifier of a first (spendable) input box of the transaction. The identifier is unique assuming the collision resistance of an underlying hash function. The amount of the tokens issued could be any number within the range $[1, 9223372036854775807]$. The weak preservation rule is followed for tokens, which requires that the total amount of any token in a transaction's outputs should be no more than the total amount of that token in the transaction's inputs (i.e., some amount of token could be burnt). In contrast, the strong reservation rule is followed for Ergs, which requires that the total amount of Ergs in the inputs and outputs must be the same.

7.2 Contract Examples

In this section, we provide some examples which demonstrate the superiority of Ergo contracts compared to Bitcoin's. The examples include betting on oracle-provided data, non-interactive mixing, atomic swaps, complementary currency, and an initial coin offering implemented on top of the Ergo blockchain.

7.2.1 An Oracle Example

Equipped with custom tokens and data inputs, we can develop a simple oracle example which also shows some design patterns that we discovered while playing with Ergo contracts. Assume that Alice and Bob want to bet on tomorrow's weather by putting money into a box that becomes spendable by Alice if tomorrow's temperature is more than 15 degrees, and spendable by Bob otherwise. To deliver the temperature into the blockchain, a trusted oracle is needed.

In contrast to Ethereum with its long-lived accounts, where a trusted oracle's identifier is usually known in advance, delivering data with one-time boxes is more tricky. For starters, a box protected by the oracle's key cannot be trusted, as anyone can create such a box. It is possible to include signed data into a box and check the oracle's signature in the contract (we have such an example), but this is quite involved. Instead, a solution with custom tokens is very simple.

Firstly, a token identifying the oracle should be issued. In the simplest case, the amount of this token could be one. We call such a token *a singleton token*. The oracle creates a box containing this token along with its data (i.e., the temperature) in register R_4 and the UNIX epoch time in register R_5. In order to update the temperature, the oracle destroys this box and creates a new one with the updated temperature.

Assume that Alice and Bob know the oracle's token identifier in advance. With this knowledge, they can jointly create a box with a contract that requires first (read-only) data input to contain the oracle's token. The contract extracts the temperature and time from the data input and decides who gets the payout. The code is as simple as following:

Archived copy · cryp2nova.com

Algorithm 3 Oracle Contract Example

```
1: val dataInput = CONTEXT.dataInputs(0)
2: val inReg = dataInput.R4[Long].get
3: val inTime = dataInput.R5[Long].get
4: val inToken = dataInput.tokens(0)._1 == tokenId
5: val okContractLogic = (inTime > 1556089223) &&
6:           ((inReg > 15L && pkA) || (inReg ≤ 15L && pkB))
7: inToken && okContractLogic
```

This contract shows how a singleton token could be used for authentication. As a possible alternative, the oracle can put the time and temperature into a box along with a signature on this data. However, this requires signature verification, which is more complex and expensive compared to the singleton token approach. Also, the contract shows how read-only data inputs could be useful for contracts which need to access data stored in some other box in the state. Without data inputs, an oracle must issue one spendable box for each pair of Alice and Bob. With data inputs, the oracle issues only a single box.

7.2.2 A Mixing Example

Privacy is important for a digital currency but implementing it can be costly or require a trusted setup. Thus, it is desirable to find a cheaper way for coin mixing. As a first step towards that, we offer a non-interactive mixing protocol between two users Alice and Bob that works as follows:

1. Alice creates a box which demands the spending transaction to satisfy certain conditions. After that, Alice only listens to the blockchain; no interaction with Bob is needed.

2. Bob creates a transaction spending Alice's box along with one of his own to generate two outputs having identical scripts but different data. Each of Alice and Bob may spend only one of the two outputs but an observer decide which output belongs to whom because they look indistinguishable.

For simplicity, we do not consider fee in the example. The idea of mixing is similar to non-interactive Diffie-Hellman key exchange. First, Alice generates a secret value x (a huge number) and publishes the corresponding public value $gX = g^x$. She requires Bob to generate a secret number y, and to include into each output two values c_1, c_2, where one value is equal to g^y and the other is equal to g^{xy}. Bob uses a random coin to choose meanings for $\{c_1, c_2\}$. Without access to the secrets, an external observer cannot guess with probability better than $\frac{1}{2}$ whether c_1 is equal to g^y or to g^{xy}. This is assuming that the cryptographic primitive we use has a certain property, that the Decision Diffie-Hellman (DDH) problem is hard. To destroy an output box, a proof should be given that either y is known such that $c_2 = g^y$, or x is known such that $c_2 = c_1^x$. The contract of Alice's box checks that c_1 and c_2 are well-formed. The code snippets for Alice's coin and the mixing transaction's output are provided

Archived copy · cryp2nova.com

in Algorithms 4 and 5 respectively. Since ErgoScript currently doesn't have support for proving knowledge of some x such that $c_2 = {c_1}^x$ for arbitrary c_1, we will prove a slightly longer statement that is supported, namely, proving knowledge of x such that $gX = g^x$ and $c_2 = {c_1}^x$. This is called proveDHTuple.

Algorithm 4 Alice's Input Script

```
1: val c1 = OUTPUTS(0).R4[GroupElement].get
2: val c2 = OUTPUTS(0).R5[GroupElement].get
3:
4: OUTPUTS.size == 2 &&
5: OUTPUTS(0).value == SELF.value &&
6: OUTPUTS(1).value == SELF.value &&
7: blake2b256(OUTPUTS(0).propositionBytes) == fullMixScriptHash &&
8: blake2b256(OUTPUTS(1).propositionBytes) == fullMixScriptHash &&
9: OUTPUTS(1).R4[GroupElement].get == c2 &&
10: OUTPUTS(1).R5[GroupElement].get == c1 && {
11:       proveDHTuple(g, gX, c1, c2) ||
12:       proveDHTuple(g, gX, c2, c1)
13: }
```

Algorithm 5 Mixing Transaction Output Script

```
1: val c1 = SELF.R4[GroupElement].get
2: val c2 = SELF.R5[GroupElement].get
3: proveDlog(c2) || // either c2 is g^y
4: proveDHTuple(g, c1, gX, c2) // or c2 is u^y = g^xy
```

We refer the reader to [37] for a proof of indistinguishability of the outputs and details on why Alice and Bob can spend only their respective coins.

7.2.3 More Examples

In this section, we briefly shed light on a few more examples along with links to the documents providing the details and code.

Atomic Swap Cross-chain atomic swap between Ergo and any blockchain that supports payment to either SHA-256 or Blake2b-256 hash preimages and time-locks can be done in a similar way to that proposed for Bitcoin [38]. An Ergo alternative implementation is provided in [36]. As Ergo also has custom tokens, atomic exchange on the single Ergo blockchain (Erg-to-token or token-to-token) is also possible. An implementation for this can also be found in [36].

Crowdfunding We consider the simplest crowdfunding scenario. In this example, a crowdfunding project with a known public key is considered successful

Archived copy · cryp2nova.com

if it can collect unspent outputs with a total value not less than a certain amount before a certain height. A project backer creates an output box protected by the following statement: the box can be spent if the spending transaction has the first output box protected by the project's key and amount no less than the target amount. Then the project can collect (in a single transaction) the biggest backer output boxes with a total value not less than the target amount (it is possible to collect up to 22,000 outputs, which is enough even for a big crowdfunding campaign). For the remaining outputs, it is possible to construct follow-up transactions. The code can be found in [36].

The Local Exchange Trading System Here we briefly demonstrate a Local Exchange Trading System (LETS) in Ergo. In such a system, a member of a community may issue community currency via personal debt. For example, if Alice with zero balance is buying something for 5 community tokens from Bob, whose balance is zero as well, her balance after the trade would be -5 tokens, and Bob's balance would be 5 tokens. Then Bob can buy something using his 5 tokens, for example, from Carol. Usually, in such systems, there is a limit on negative balances (to avoid free-riding).

Since a digital community is vulnerable to Sybil attacks [39], some mechanism is needed to prevent such attacks where Sybil nodes create debts. The simplest solution is to use a committee of trusted managers that approve new members of the community. A trust-less but more complex solution is to use security deposits made in Ergs. For simplicity, we consider the approach with the committee here.

This example contains two interacting contracts. A *management contract* maintains a list of community members, and a new member can be added if some management condition is satisfied (for example, a threshold signature is provided). A new member is associated with a box containing a token that identifies the member. This box, which contains the *member contract*, is protected by a special exchange script that requires the spending transaction to do a fair exchange. We skip the corresponding code, which can be found in a separate article [40].

What this contract shows, in contrast to the previous example, is that instead of storing the members list, only a short digest of an authenticated AVL+ tree can be included in the box. This allows a reduction in storage requirements for the state. A transaction doing lookup or modification of the member list should provide a proof for AVL+ tree lookup or modification operations. Thus, saving space in the state storage leads to bigger transactions, but this scalability problem is easier to solve.

Initial Coin Offering We discuss an Initial Coin Offering (ICO) example that shows how multi-stage contracts can be created in Ergo. Like most ICOs, our example has three stages. In the first stage, the project raises money in Ergs. In the second stage, the project issues a new token, whose amount equals the number of nanoErgs raised in the first stage. In the third stage, the investors

Archived copy · cryp2nova.com

can withdraw issued tokens.

Note that the first and third stages have many transactions on the blockchain, while a single transaction is enough for the second stage. Similar to the previous example, the ICO contract uses an AVL+ tree to store the list of (investor, amount) pairs. The complete code is available at [41].

More Examples We have even more examples of Ergo applications in [36, 37]. These examples include time-controlled emission, cold wallets contracts, rock-paper-scissors game, and many others.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] Bitcoin's price surpasses $18,000 level, market cap now higher than visa's. [Online]. Available: https://cointelegraph.com/news/bitcoins-price-surpasses-18000-level-market-cap-now-higher-than-visas

[3] A. Chepurnoy, V. Kharin, and D. Meshkov, "A systematic approach to cryptocurrency fees," *IACR Cryptology ePrint Archive*, vol. 2018, p. 78, 2018.

[4] Skyrocketing fees are fundamentally changing bitcoin. [Online]. Available: https://arstechnica.com/tech-policy/2017/12/bitcoin-fees-rising-high/

[5] N. Szabo, "Smart contracts," *Unpublished manuscript*, 1994.

[6] J. Zahnentferner, "Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies." *IACR Cryptology ePrint Archive*, vol. 2018, p. 262, 2018.

[7] I accidentally killed it: Parity wallet bug locks $150 million in ether. [Online]. Available: https://www.ccn.com/i-accidentally-killed-it-parity-wallet-bug-locks-150-million-in-ether

[8] A. Chepurnoy, V. Kharin, and D. Meshkov, "Self-reproducing coins as universal turing machine," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 57–64.

[9] M. Jansen, F. Hdhili, R. Gouiaa, and Z. Qasem, "Do smart contract languages need to be turing complete?" 2019. [Online]. Available: https://www.researchgate.net/publication/332072371_Do_Smart_Contract_Languages_Need_to_be_Turing_Complete/download

[10] Very slow syncing on hard drive. [Online]. Available: https://github.com/ethereum/go-ethereum/issues/14895

Archived copy · cryp2nova.com

[11] Why is my node synchronization stuck/extremely slow at block 2,306,843? [Online]. Available: https://ethereum.stackexchange.com/questions/9883/why-is-my-node-synchronization-stuck-extremely-slow-at-block-2-306-843

[12] L. Reyzin, D. Meshkov, A. Chepurnoy, and S. Ivanov, "Improving authenticated dynamic dictionaries, with applications to cryptocurrencies," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 376–392.

[13] L. Goodman, "Tezos — a self-amending crypto-ledger white paper," *URL: https://www. tezos. com/static/papers/white_paper. pdf*, 2014.

[14] A. Biryukov and D. Khovratovich, "Equihash: Asymmetric proof-of-work based on the generalized birthday problem," *Ledger*, vol. 2, pp. 1–30, 2017.

[15] Ethash. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethash/6e97c9cea49605264c6f4d1dc9e1939b1f89a5a3

[16] A. Chepurnoy, V. Kharin, and D. Meshkov, "Autolykos: The ergo platform pow puzzle," 2019. [Online]. Available: https://docs.ergoplatform.com/ErgoPow.pdf

[17] Autolykos gpu miner. [Online]. Available: https://github.com/ergoplatform/Autolykos-GPU-miner

[18] D. Meshkov, A. Chepurnoy, and M. Jansen, "Short paper: Revisiting difficulty control for blockchain systems," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 429–436.

[19] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, Tech. Rep., 2018.

[20] Ethereum's blockchain is once again feeling the 'difficulty bomb' effect. [Online]. Available: https://www.coindesk.com/ethereum-blockchain-feeling-the-difficulty-bomb-effect

[21] E. Heilman, N. Narula, G. Tanzer, J. Lovejoy, M. Colavita, M. Virza, and T. Dryja, "Cryptanalysis of curl-p and other attacks on the iota cryptocurrency."

[22] G. De Roode, I. Ullah, and P. J. Havinga, "How to break iota heart by replaying?" in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–7.

[23] Iota vulnerability report: Cryptanalysis of the curl hash function enabling practical signature forgery attacks on the iota cryptocurrency. [Online]. Available: https://github.com/mit-dci/tangled-curl/blob/master/vuln-iota.md

Archived copy · cryp2nova.com

[24] A. Chepurnoy and M. Rathee, "Checking laws of the blockchain with property-based testing," in *Blockchain Oriented Software Engineering (IWBOSE), 2018 International Workshop on*. IEEE, 2018, pp. 40–47.

[25] T. Duong, A. Chepurnoy, and H.-S. Zhou, "Multi-mode cryptocurrency systems," in *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*. ACM, 2018, pp. 35–46.

[26] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," Cryptology ePrint Archive, Report 2017/963, 2017. Accessed: 2017-10-03, Tech. Rep., 2017.

[27] L. Luu, B. Buenz, and M. Zamani, "Flyclient super light client for cryptocurrencies," *IACR Cryptology ePrint Archive*, 2019. [Online]. Available: https://eprint.iacr.org/2019/226

[28] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Another coin bites the dust: an analysis of dust in utxo-based cryptocurrencies," *Royal Society open science*, vol. 6, no. 1, p. 180817, 2019.

[29] Ethereum network attacker's ip address is traceable. [Online]. Available: https://www.bokconsulting.com.au/blog/ethereum-network-attackers-ip-address-is-traceable/

[30] A. Chepurnoy and D. Meshkov, "On space-scarce economy in blockchain systems." *IACR Cryptology ePrint Archive*, vol. 2017, p. 644, 2017.

[31] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, "On the instability of bitcoin without the block reward," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 154–167.

[32] E. Krause, "A fifth of all bitcoin is missing. these crypto hunters can help," 2018.

[33] 51% attack. [Online]. Available: https://en.bitcoinwiki.org/wiki/51%25_attack

[34] Script of the ergo treasury box. [Online]. Available: https://github.com/ScorexFoundation/sigmastate-interpreter/blob/1b7b5a69035fc384b47c18ccf42b87dd95bbcf12/src/main/scala/org/ergoplatform/ErgoScriptPredef.scala#L118

[35] Script of the ergo emission box. [Online]. Available: https://github.com/ScorexFoundation/sigmastate-interpreter/blob/1b7b5a69035fc384b47c18ccf42b87dd95bbcf12/src/main/scala/org/ergoplatform/ErgoScriptPredef.scala#L74

Archived copy · cryp2nova.com

[36] Ergoscript, a cryptocurrency scripting language supporting noninteractive zero-knowledge proofs. [Online]. Available: https://docs.ergoplatform.com/ErgoScript.pdf

[37] Advanced ergoscript tutorial. [Online]. Available: https://docs.ergoplatform.com/sigmastate_protocols.pdf

[38] T. Nolan, "Alt chains and atomic transfers," 2013. [Online]. Available: https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949

[39] Sybil attack. [Online]. Available: https://en.wikipedia.org/wiki/Sybil_attack

[40] A local exchange trading system on top of ergo. [Online]. Available: https://ergoplatform.org/blog/2019_04_22-lets/

[41] An ico example on top of ergo. [Online]. Available: https://ergoplatform.org/blog/2019_04_10-ico-example/

CRYP2NOVA

Archived copy · cryp2nova.com