Ergo: A Resilient Platform For Contractual Money

Ergo Developers

This is an early draft of the whitepaper

Abstract

In this work, we present Ergo, a new flexible and secure decentralized blockchain protocol. Ergo is designed to be a platform for applications with the main focus to provide an efficient, secure and easy way to implement financial contracts.

To achieve this goal, Ergo includes various technical and economic improvements to existed 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. It specifies conditions under which a coin can be used: who can spend them, when, under what external conditions, to whom, and so on.

Extended support of light nodes makes Ergo friendly for end users and allows to run contracts on commodity hardware without any trust. To be useful in the long-term, Ergo is following survivability approach — it uses widely-researched solutions that are not going to result in security issues in the future, while also prevents performance degradation over time with the new economic model.

Finally, Ergo is a self-amendable protocol, that allows to absorb new ideas and improve itself in the future.

1 Introduction

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

However, this resilience does not come for free and has not yet been proven by existing systems in the long-term 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.

Besides network utilization, transaction processing requires to utilize computational resources, especially if the transactional language is flexible enough. Finally, blockchain participants should keep quite a significant amount of data in their local storages and the storage requirements are growing fast. Thus transaction processing utilize various resources of thousands of computers all over the world and these resources consumption is paid by regular users via transaction fees [3]. Despite the generous block reward subsidy in some existing systems, these fees may be very high [4] and ten years later blockchain technology is still mainly used in financial applications, where the advantage of high security outweighs the disadvantage of high transaction costs.

Besides of a plain currency usage, most of the blockchains are used to build decentralized applications on top of them. Such applications utilize the ability to write smart contracts [5], that implements their logic by means of blockchain-specific programming language. For now, there are two main approaches to write smart contracts [6]: UTXO-based (e.g., Bitcoin) and account-based (e.g., Ethereum). Account-based cryptocurrencies, such as Ethereum, introduce special contract accounts controlled by code, that may be invoked by incoming transactions. This approach allows performing arbitrary computations, however, implementation of complex coins spending conditions lead to bugs like $150 million loss in 2017 in Ethereum from simple multi-signature contract [7]. In UTXO-based cryptocurrencies, every coin has a script associated with it, and to spend the coin, one should satisfy script conditions. Implementation of coins protecting conditions is much easier in UTXO model, while arbitrary Turing-complete logic become quite complicated (e.g., implementation of a trivial Turing-complete system become quite complicated [8]). Ergo is based on UTXO model as far as it is more convenient to implement financial applications covering the overwhelming majority of public blockchain use-cases.

While the contractual component is attractive in for building decentralized applications, it is also essential that the blockchain will survive in the long term. For now, the whole area is young, and most of the application-oriented blockchain platforms exist just for several years, while they already have known problems with performance degradation over time [**?**] and their long-term survivability is questionable. And even older UTXO money-oriented blockchains have not yet been proven fully resilient in the long-run under changing conditions as we only have 10-years of blockchain history to this point. This problem led to concepts of light nodes with minimum storage requirements [9], storage rent fee component that prevents bloating of full-node requirements [3], self-amendable protocols that can adapt to the changing environment and improve themselves without trusted parties [10], and more. What is needed is the 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.

2 Ergo Vision

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 during the Ergo live 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) which absence or malicious behavior may affect the security of the network should be avoided. If any of these parties will appear during Ergo live, the community should consider ways how to decrease their impact level.
- *Created for regular people.* Ergo is the platform for ordinary people, and their interests should not be infringed upon in favor of big parties. In particular, that means that 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 any applications, but the main focus is to provide an efficient, secure and easy way to implement financial contracts.
- *Long terms 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. As far as Ergo is oriented to be 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 responsibilit y 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 PoW protocol called the Autolykos Consensus

Protocol further described below. Despite extensive research of possible alternatives to the original Proof-of-Work (PoW) protocol with the longest chain rule, is still in demand due to simplicity, high-security guarantees, and friendliness to light clients. However, a decade of extensive testing revealed several problems of the original one-CPU-one-vote idea.

First known problem of PoW is that development of specialized hardware (ASIC) 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 the 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 [11, 12].

The second known threat to a PoW network decentralization is that even big miners trend 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 were implementer before Ergo.

Ergo PoW protocol, Autolykos [13], is the first consensus protocol, that is both memory-hard and pool-resistant. Autolykos is based on one list k-sum problem: miner should find $k = 32$ elements from the pre-defined list R of size $N = 2^{26}$ (which have 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, we require a set of element indexes J 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 algorithm 1 to create a valid block.

Algorithm 1 Block mining

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

Note that although the mining process utilizes private keys, the solution itself only contains public keys. Solution verification can be performed by Alg. 2.

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, as soon as a miner that found a correct solution can always try to steal the block reward as far as he knows the secret key sk. On the other hand, it is secure to reveal a single solution, as soon as it only operates with public keys and reveals only one linear relation between 2 secrets sk, w.

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

Calculating the list R is also quite a heavy computational task: our initial implementation [14] 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 milliseconds (about 50 ms on Nvidia GTX 1070).

Target parameter b is built-in into the puzzle itself and is adjusted to the current network hash rate via difficulty adjustment algorithm [15] to keep time interval between block close to 2 minutes. This algorithm is trying to predict the hash rate of an upcoming 1024 block length epoch based on data from the previous 8 epochs via well-known linear least squares method, to predict it better than the usual difficulty adjustment algorithm while also makes possible "coin-hopping" attacks less profitable.

4 Ergo state

To check a new transaction, a cryptocurrency client is not using the ledger (all the transactions happened before the transaction), rather, it is using ledger state snapshot got from the history. In Bitcoin Core reference implementation, this snapshot is about active one-time coins, and a transaction is destroying some coins and also creating new ones. In Ethereum, the snapshot is about long-living accounts, where an account is controlled whether by a human or executable code; a transaction then is modifying monetary balance and internal memory of some accounts. Also, the representation of the snapshot in Ethereum (unlike Bitcoin) is fixed by the protocol, as authenticating digest of the snapshot is written into a block header (thus, in order to have full security guarantees, a client needs to build the same snapshot as a miner).

Ergo is representing the snapshot in the form of one-time coins, like Bitcoin. The difference is, in addition to monetary value and protecting script, an Ergo coin contains additional registers with arbitrary data, thus we use term *box* instead of *coin*. Using one-time immutable objects is the easiest and safest solution for replay and reordering attacks. Also, it it is easier to process transactions in parallel when they are not modifying state of objects they are accessing. Also, with one-time coins, it seems it is easier to build fully stateless clients [16], however, research in this area is still in the initial stage. A major criticism for one-time coins says that this model is not suitable for non-trivial applications, but Ergo has overcome such problems, and we have many non-trivial prototype applications built on top of the Ergo Platform.

Like in Ethereum, the ledger snapshot representation (in the form of boxes not destroyed by previous transactions) is fixed by the Ergo protocol. In details, a miner should maintain a Merkle-tree like authenticated data structure built on top of UTXO set, and include short digest (just 33 bytes) corresponding to UTXO set after application of a block into a header of the block. However, static Merkle trees are not suitable for evolving datasets, so we are using authenticated AVL+ trees as described in [9]. We provide access to this datastructure via our transaction language as well.

5 Resiliency and Survivability

Being a platform for contractual money, Ergo should also support long-term contracts for a period of a person's life. At the same time, even young existing smart contract platforms are experiencing issues with performance degradation and adaptability to external conditions. This leads to a situation where a cryptocurrency depends on a small group of developers that should provide a fixing hard-fork, or the cryptocurrency won't survive.

The first common survivability issue is that in pursuit of popularity blockchain developers implement ad-hoc solutions without proper preliminary research and testing. Such solutions inevitably lead to bugs, hasty bug fixes, fixes of bug fixes and so on, making the network even less secure. Ergo approach here is to use stable well-tested solutions, even if that leads to slower short-term innovations. Most of Ergo solutions are formalized in papers presented at peer-reviewed conferences and have been widely discussed in the community.

A second problem decentralization and thus survivability faces is lack of secure trustless light clients. Ergo is trying to fix known problems of blockchain technology without creating new problems. As far as Ergo is a PoW blockchain, it easily allows extracting a small header from the block content. Header alone allows for validation of the work done on it, while headers-chain is enough for best chain selection and synchronization with the network. Headers-chain is much smaller than the full blockchain, however, it is still growing linearly with time. Hopefully, modern research of light clients [17, 18] provide a way to synchronize with the network by downloading even smaller amount of data, unlocking the ability to use the network without any trust from low-end hardware like mobile phones. Also, Ergo uses authenticated state [9] and for transactions included a client may download a proof of their correctness. The proof is generated by a block miner and allows to check all the state transitions happened within the block, namely, that transaction inputs not being spent before the transaction, and that the transaction outputs has been indeed added to the state. 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 see a third potential problem in that although support of light clients solves problems of Ergo users, it does not solve problems of Ergo miners that still should keep the whole state for efficient transaction validation. In existing blockchain systems, users can put arbitrary data to this state forever, creating a lot of dust in it and increasing its size over time [19]. Big state size leads to serious security issues when the state does not fit in random-access memory, an adversary may generate transactions which validation become very slow due to required random access to miners storage leading to DoS attack like an attack to the Ethereum network in 2016 [**?**]. Moreover the community's fear of these attacks and the problem of "state bloat" without any sort of compensation to miners and users holding the state may have prevented scaling solutions that otherwise could have been implemented such as somewhat 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 moved, a miner may charge a small fee for every byte kept in the state. This idea is similar to regular cloud storage services however, it was only proposed quite recently for cryptocurrencies [20] and has several important consequences. First, Ergo mining will always be stable, unlike Bitcoin and other PoW currencies, in which mining may become unstable after emission done [21]. Second, state size growth becomes controllable and predictable reducing hardware requirements for Ergo miners. Third, by collecting storage fees from outdated boxes, miners return coins to circulation preventing a steady decrease of circulating supply due to lost keys [22]. All these effects should support Ergo long-term survivability, both technically and economically.

A fourth vital challenge to survivability is that the environment changes, so the network should adapt to ever changing hardware infrastructure, ideas that may improve security or scalability emerge over time, use-cases evolution, and so on. If all the rules are fixed without any ability to change them in a decentralized manner, even simple constant change may lead to hot debates and community splits, e.g., discussion of a block size limit in Bitcoin led to the network split 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 miners voting. At the beginning of a 1024-blocks voting epoch a miner is proposing changes (up to 2 parameters, e.g., to increase block size and to decrease storage fee factor) and during the rest of the epoch miners vote to whether approve the changes or not. If the majority of votes within an epoch are supporting a change, a new value of the parameter should be written into the extension section of the first block of the next epoch, and the network starts to use this update parameter value during block mining and validation.

To absorb more fundamental changes, Ergo is following the approach of soft-forkability that allows to change protocol significantly with keeping old nodes operating. At the beginning of an epoch, a miner can also propose to vote for a fundamental change (e.g., to add a new instruction to ErgoScript), describing affected validation rules. Voting for such breaking changes continues for 32,768 blocks and requires for at least 90% of "Yes" votes to be accepted. Once being accepted, 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 current validation rules.

6 Ergo's Native Token

In this section, we are going to provide some reflections on the nature of the native Ergo token. For starters, we note that any currency is about three main functions: a medium of exchange, a unit of account, and a store of value.

Bitcoin, being historically the first digital scarce asset, is perfect as store-of-value. It is even better than gold under certain assumptions (such as SHA-256 hash function not being broken, and majority of miners are not willing to destroy the Bitcoin), as emission is limited and known in advance. However, being perfect store-of-value also means to be not so good as medium-of-exchange. In particular, if one knows that Bitcoin is indeed the best tool to store value in the long term, he would use fiat whenever possible in order to collect more bitcoins.

On the other hand, Ethereum is not just a currency, but a utility token used to pay for computations over the "decentralized world computer" (or "fully replicated programmable calculator"). However, Ethereum is not good as store-of-value, as emission in endless, and, historically, can be changed easily by the community core along with critical system assumptions.

Ergo combines best from these two top blockchains. Emission is predefined and limited, more, it will be finished within just ten years. The system assumptions are set in stone with precisely defined *social contract* 2. Also, Ergo is a utility token used to pay for storage rent. This storage rent is making the system more stable. Last, Ergo is suitable to build monetary systems on top of it with properties different from Ergo native token itself. However, participating in such systems would require to use Ergo native token as well in order to pay for storage rent.

6.1 Currency And Emission

The native currency of Ergo platform is **Erg** token, which unique property is that it is the only currency to pay for storage rent in Ergo (see Section 5 for more details). One **Erg** token is divisible to up to 10^9 smallest units, **nanoErg**s.

All **Erg** tokens that will ever circulate in the system are presented in the initial state and are divided into 3 parts (boxes):

- *No premine proof.* This box contains exactly one **Erg** and is protected by the script that is preventing it from spending by anyone. Thus, it is a long-lived box that will stay in the system until 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 contains latest titles from media (The Guardian, Vedomosti, Xinhua) as well as latest block identifiers from Bitcoin and Ethereum. Thus, Ergo mining could not be started before certain events in the real world and the cryptocurrency space.

- *Treasury.* This box contain 4,330,791.5 **Erg** that will be used to fund Ergo development. Its protecting script [?] 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 **Erg** will be released every block, during blocks 525,600-590,399 (3 months) 4.5 **Erg** will be released every block, and finally, during blocks 590,400-655,199 (3 months) 1.5 **Erg** will be released every block. This rule ensures the presence of funds for Ergo development for at least 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 that spending transaction should be signed by at least 2 of 3 secret keys that are under control of 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 are going to be used to cover 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 **Erg** that will be collected by block miners as rewards for their work. Its protecting script [?] requires that spending transaction should have exactly two outputs with the following properties:
  - the first output should be protected by the same script and number of **Erg** in it should equal to the remaining miners' reward. During blocks 1 - 525,599 (2 years) miner will be able to collect 67.5 **Erg** from this box, during blocks 525,600 - 590,399 (3 month) miner will be able to collect 66 **Erg** and after that block reward will be reduced by 3 **Erg** every 64,800 blocks (3 months) until it reaches zero at block 2,080,799.
  - 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 these rules results in the following curve of the number of coins in circulation with time:

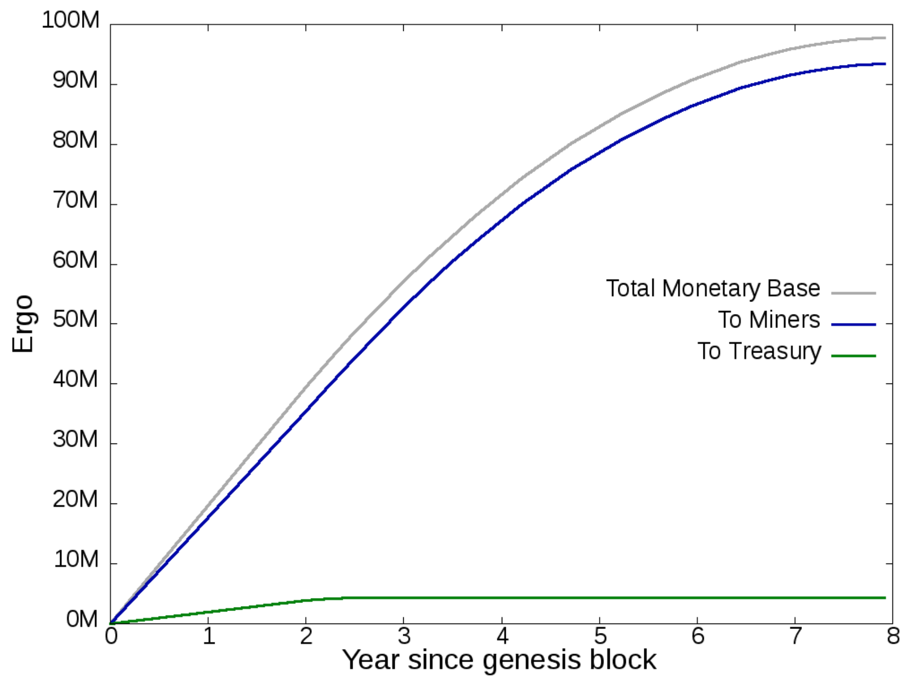

Figure 1: Ergo emission curve

7 Contractual Money

In our opinion, the overwhelming majority of use-cases for a public blockchain are about financial applications, even in case of a platform which claims to be a general-purpose decentralized world computer. For example, if an oracle is writing down non-financial data into a blockchain (such as temperature), this data is usually to be used further in a financial contract. Another trivial observation we have made is that many applications are using digital tokens with mechanics different from a native one.

The Ergo Platform is offering an application developer custom tokens (which are first-class citizens) and a domain-specific language for box protecting condition in order to implement flexible and secure financial applications. Ergo applications are defined in terms of protecting scripts built into boxes also containing data possibly involved into execution. We coin the term *contractual money* for Ergs and secondary tokens which usage is bounded by a contract, so all the tokens on the platform in existence.

Any box with its contents (Ergs, tokens, data) is bounded by a contract. However, we can distinguish Ergs in existence as ones which could easily change their contracts from Ergs which are bounded by contracts in the sense that a box with contractual Ergs is demanding from a spending transaction to create boxes with some properties. We will refer to the former as to ordinary (or cleared, or free) Ergs, and to the latter as to the bounded Ergs. Similarly, we can define bounded tokens. Both free and bounded Ergs are contractual, and in some cases hybrids are possible, for example, for one public key it could be a bounding contract, and for another public key not.

For example, if a box is protected just by a public key (so providing a signature against a spending transaction is enough in order to destroy the box), a public key owner may create an arbitrary box replacing the one being protected by the public key, thus the Ergs within the box are free to change the contract. In contrast, imagine a box "B" which is protected by combination of a public key and also condition which demands a spending transaction to create an output box which guarding script hash is equal to "rBMUEMuPQUx3GzgFZSsHmLMBouLabNZ4HcERm4N" (in Base58 encoding), and Ergs value of the output should equal to the value of the original box. In this case, box value is bounded by the contract, thus Ergs in the box are bounded Ergs.

7.1 Prelimiaries For Ergo Contracts

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

However, in most cases, a developer is unlikely to develop box contracts in terms of graphs. Instead, he would likely using a high-level language, and we provide one called ErgoScript. Writing scripts in this language is easy, for example, for a one-out-of-two signature, protecting script would be $pk_1 \&\& pk_2$, which means "prove knowledge of a secret key corresponding to the public key pk_1 and knowledge of a secret key corresponding to pk_2". We have two separate documents which are helping to develop contracts with ErgoScript, the "ErgoScript Tutorial" [23] and "Advanced ErgoScript Tutorial" [24]. Thus below we are not going to dive into developing contracts with ErgoScript, rather, we are going to provide a couple of motivation examples in following sections.

More features of Ergo are shaping contracting possibilities:

- Data Inputs: a box could be not only destroyed by a transaction, but is also could be only read, in the latter case we refer to the box as to *data input* of the transaction. Thus a transaction is getting two box sets as its arguments, inputs and data inputs, and produces a box set named *outputs*. Data inputs are useful for oracle applications and interacting contracts.

- Custom tokens: a transaction can carry many tokens, if only estimated complexity for processing them is not exceeding a limit. A transaction is also able to issue a token, but only one, and with a unique (and cryptographically strong against finding a collision) identifier, which is equal to identifier of a first (spendable) input box of the transaction. The amount of the tokens issued could be any number within the [1, 9223372036854775807] range. For the tokens, the weak preservation rule is defined, which is demanding total amount for a token in transaction outputs should be no more than total amount for the token in transaction inputs (thus some amount of token could be burnt). In contrast, for Ergs a preservation rule is strong, thus total Ergs amount for inputs should be equal to total Ergs amount for outputs.

7.2 Contract Examples

In this section we provide some examples which are clearly show superiority of Ergo contracts in comparison with Bitcoin. The examples are including a bet on oracle-provided data, non-interactive mixing, and a complementary currency living 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, also showing on the way some design patterns that we have discovered. Assume that Alice and Bob are going to bet on weather tomorrow. For that, they are putting money into a box, which is spendable by Alice if the temperature is more than 15 degrees, and is spendable by Bob otherwise. To deliver the temperature into the blockchain a trusted oracle is needed.

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

In the first place, a token identifying the oracle should be issued. In the simplest case, the amount for the token could be equal to one. We call such a token *a singleton token*. Then the oracle is creating a box containing the token and also data, namely, temperature in the register number four ($R4$), and time (in seconds since the beginning of the UNIX epoch) in the register number five($R5$). In order to update temperature, an oracle is destroying the box and creating a new one with updated temperature.

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

Algorithm 3 Oracle Contract Example

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

This contract shows how a singleton token could be used for authentification. As a possible alternative, an oracle can put signed time and temperature into a box along with a signature for the data published. However, this requires signature checking in a contract which is using an oracle box, which is more complex and expensive, in comparison with the singleton token approach. Also the contract shows how read-only data inputs could be useful for contracts which do need access to data in some box in the state. Without data inputs, an oracle needs to issue a spendable box for every pair of Alice and Bob. With data inputs support, the oracle is issuing only a single box per arbitrary number of users.

7.2.2 A Mixing Example

Privacy is important for a digital currency, but implementing it in a protocol could be costly or require a trusted setup. Thus we are looking for ways to do coin mixing via cheap enough applications. As a first step towards that, we offer an application for non-interactive coin mixing, which is working in the following case:

1. Alice creates a box which demands any Bob's box to satisfy certain conditions in order to be mixed with the coin of Alice. After that, Alice only listens to the blockchain, no any interaction with Bob is needed.

2. Bob is creating a box and then a spending transaction which has boxes of Alice and Bob as inputs, and creates two outputs with the same script, but both Alice and Bob may spend only one box out of the two. An external observer can not decide which output is spent by whom, as output boxes are indistinguishable.

For simplicity, we are not considering fees in the example. The idea of mixing is similar to non-interactive Diffie-Hellman key exchange. First, Alice creates a secret value x (a huge number) and publishing a corresponding public value $gX = g^x$. She demands from Bob to generate a secret number y, and to include into each output two values $c1$, $c2$, where one value is equal to g^y and another is equal to $g^{x \cdot y}$. We assume that a Bob is using a random coin to choose meanings for $\{c_1, c_2\}$. With no access to the secret information, an external observer can not guess whether, for example, c_1 is about g^y or $g^{x \cdot y}$, with probability above $\frac{1}{2}$ (as a cryptographic primitive we are using has a certain property, namely, the hardness of Diffie-Hellman decisional problem). To destroy an output box, a proof should be given that for c_2 whether y is known, such that $c_2 = g^y$, or $c_2 = g^{x \cdot y}$ has the same exponent x against $c1 = g^y$ as g^x against g. The contract of the Alice's box is checking that $c1$ and $c2$ are well-formed. The code snippets for the Alice's coin and a mixing transaction output are provided in Algorithm 4 and Algorithm 5, respectively.

Algorithm 4 Alice's Input Script

```
val c1 = OUTPUTS(0).R4[GroupElement].get
val c2 = OUTPUTS(0).R5[GroupElement].get

OUTPUTS.size == 2 &&
OUTPUTS(0).value == SELF.value &&
OUTPUTS(1).value == SELF.value &&
blake2b256(OUTPUTS(0).propositionBytes) == fullMixScriptHash &&
blake2b256(OUTPUTS(1).propositionBytes) == fullMixScriptHash &&
OUTPUTS(1).R4[GroupElement].get == c2 &&
OUTPUTS(1).R5[GroupElement].get == c1 && {
    proveDHTuple(g, gX, c1, c2) ||
    proveDHTuple(g, gX, c2, c1)
}
```

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 $u^y = g^{x \cdot y}$
```

We refer to the [24] for a proof of outputs indistinguishability and details on why Alice and Bob can spend only their respective coins.

7.2.3 More Examples

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

Atomic Swap Cross-chain atomic swap Ergo and another blockchain (supporting a payment to SHA-256 or Blake2b-256 hash preimage and a time-lock) could be done in the same way as proposed for Bitcoin, see initial protocol for Bitcoin in [25], its Ergo alternative is provided in [23]. As Ergo has custom tokens, atomic exchange on a single block chain (Erg-to-token or token-to-token) is also possible, a solution for that can be found [23].

Crowdfunding We consider a simplest crowdfunding scenario. In the example, a crowdfunding project with a known public key is considered successful if it can collect unspent outputs with 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 a spending transaction is having its first output box protected by the project's key, and the output should have no less than the target amount. Then the project can collect biggest backer output boxes with total value not less than the amount with a single transaction (it is possible to collect up to 22,000 outputs in Bitcoin, which is enough even for a big crowdfunding campaign). For remaining small outputs over the amount, it is possible to construct follow-up transactions. The code can be found in [23].

The Local Exchange Trading System Here we briefly overview a local exchange trading system implementation. 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, which balance is about 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 for his 5 tokens, for example, from Carol. Usually, in such systems, there is a limit for negative balance (to avoid free-riding).

As digital community could be vulnerable to sybil attacks (which allow to do free-riding again), thus some mechanism is needed in order to prevent the creation of sybils creating debts. We consider two solutions, namely, a committee of trusted managers approving new members of the community, or security deposits made in Ergs. For simplicity, we consider the approach with the committee here.

This example is about two interacting contracts then. A management contract is about maintaining a community members list, and a new member could be added if management condition satisfied (for example, a threshold signature is provided). A new member is associated with a box which contains a token identifying the member. The user box is protected by a special exchange script requiring to do a fair exchange only for a spending transaction. We skip the corresponding code, but it could be found in a separate article [26].

Initial Coin Offering

Even More Examples We have more examples of Ergo applications, such as crowdfunding, atomic swap, time-controlled emission, cold wallet, rock-paper-scissors game, initial coin offering (ICO), and so on. The examples could be found in [23, 24].

8 Conclusions

References

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

[2] Bitcoins price surpasses $18,000 level, market cap now higher than visas. [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] 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.

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

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

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

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

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

[15] 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.

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

[17] 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.

[18] 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

[19] 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.

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

[21] 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.

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

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

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

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

[26] A local exchange trading system on top of ergo. [Online]. Available: https://github.com/ergoplatform/ergo/wiki/A-Local-Exchange-Trading-System-On-Top-Of-Ergo