Trustless, Tradable, Reality-Based Contracts

Roman Brown
romanbrown54@gmail.com

December 11, 2014

Abstract

Bitcoin [1] introduced the concept and a successful implementation of a digital cryptocurrency to the world. Since then, there has been much work on Bitcoin and other technologies to implement contracts, or agreements that involve more than just a one-time transfer of money. Currently, not only are there no successful, working implementations of contracts, but none of the attempts are even taking the right approach! I describe the proper way to implement contracts and a step-by-step approach of how to augment a Cryptonote-based currency to support contracts.

1 What Contracts Are

A contract is, at its core, an agreement between two parties to transfer money based on some external event happening in combination with some simple math. For example, Alice and Bob can agree that if the Dow Jones increases by 200 points tomorrow, Alice will give Bob 50 coins; otherwise, Bob will give Alice 50 coins. Alternatively, they can agree that for every point above 200, Alice gives Bob one coin, up to a certain maximum. Or they can agree that Alice or Bob pays the difference from the previous close. Other examples include betting on sports or races.

A successful implementation of a contract must be:

1.1 Reality-based

The result of a contract — how much money gets transferred where — must be linked to an external event occurring in reality. It should not depend on one or more agents' decision or assessment of how the contract should be graded.

1.2 Trustless

Because a contract must be **reality-based**, there must necessarily be some agent or agents involved in obtaining the information of what occurred in reality and introducing it to the network so that all nodes can calculate the result of

the contract. Thus it is impossible to have *completely* trustless contracts — ultimately, the network must rely on **information-gatherers** behaving properly. A successful implementation of contracts must minimize the trust, as much as possible, of these information-gatherers. Further, the incentives must be aligned so as to make it almost impossible and also financially untenable for bad information to get accepted by the network. For example, an implementation with one or two central information-gatherers would fail to meet this criteria.

If this seems unpalatable, consider the following two points:

1.2.1 Bitcoin Already Requires Trust

Bitcoin [1], along with any cryptocurrency using proof-of-work, already relies on trusting that the majority of nodes are well-behaved. If the majority of nodes are malicious and coordinated then they can double-spend to whatever degree they like. Attempting to do contracts on Bitcoin requires even more trust, as described in section 3.1.

1.2.2 Reality Relies on Consensus

This situation accurately reflects reality. Although events do actually occur in objective reality, we rely on the consensus of the people around us in order for people to take actions in accordance with said reality. If something occurs, nobody observes it but you, and there is no physical evidence, then you will be hard-pressed to prove that it happened. If something occurs, but everyone involved colludes against you by pretending that it did not, then again you will have no luck getting the proper outcome.

Let's take a simple example of depositing money in a bank. You bring $100 to the teller for them to credit your account. You leave, and you notice that you are $100 shorter, but your account has not been credited. Perhaps it was an honest mistake, or perhaps it was malicious. You bring this to the attention of the bank. Likely, they have video cameras trained on you and on the tellers to prevent this sort of dishonesty. They can check the video evidence — a separate record of objective reality — use it to determine that you did give the money to the teller, and from there investigate what went wrong. However, if they have no video evidence, and the teller forgets or tells a lie, then it'll be your word against the teller's, without any easy way to objectively prove what happened.

Note that in the previous examples, the incentives are all aligned to prevent that situation from happening. It's in the bank's interest to hire trustworthy tellers. It's in their interest to have redundant records of what occurred, such as video cameras, in order to be able to objectively resolve these disputes. It's in the teller's interest to keep their job, which is probably more valuable than stealing $100 in the moment. You also have further recourse — if this actually happens to you, and the bank does nothing to resolve it, you can tell everybody what happened, providing bad PR to the bank and likely a loss of business. This provides additional incentives to the bank to act honorably. So, all the

incentives are aligned for everybody to be able to determine what objectively occurred, even though consensus is still required.

1.3 Tradable

Ideally, contracts would also be **tradable**. Take the example of stock market derivatives. If you buy an option or a future for a certain price, you can choose to further trade the option/future. If you have a mortgage with a bank, the bank can choose to resell your mortgage to another bank. A currency which allows for a secondary market of contracts is far superior to one that does not.

2 Why Cryptonote?

I'd like to properly implement contracts on a **Cryptonote-based currency** as opposed to any other currency. In my opinion, although Cryptonote hasn't received much interest, the technology is superior and worth building on.

2.1 Privacy

The last two years have seen a big focus on privacy in the cryptocurrency community. There have been attempts to create anonymous coins such as Dark Coin [7] with their Dark Send, as well as attempts to anonymize Bitcoin itself with CoinJoin [9] implementations such as Dark Wallet [8] and SharedCoin [17]. The major benefit of Cryptonote is that it resolves all of these issues with its superior cryptographic technology, leading to **unlinkable** and **untraceable** transactions [6]. It offers maximum confidentiality if desired, as well as the ability to be completely open if necessary.

So far all of the reviews of the cryptography have shown that the algorithms are sound, for example the review by Surae Noether [14]. To implement contracts on Cryptonote is by no means straightforward, but if implemented correctly the benefits are vast: since contracts would be treated just like any other currency, the advantages of the superior cryptography would extend to contracts as well, leading to **unlinkable**, **untracable tradable** contracts.

2.2 Lack of Interest

There have been several reasons for the lack of interest in Cryptonote, none of which ultimately invalidate using the technology as a basis.

2.2.1 Complex, Raw Codebase

The codebase itself is fairly complex, using a lot of advanced C++ features. It's difficult to process the codebase and learn its ins and outs. The code is full of inconsistent formatting and misspellings. However, in the process of developing these ideas, I've spent a lot of time working with the code base, so I am already familiar enough with it to commence development on the ideas.

2.2.2 Lack of Development

Another issue is that since the technology itself was created, there hasn't been much further development on it. None of the forks provide any new substantial new features. However, this work itself will be several significant new features, which will solve precisely this problem.

3 Current Implementations of Contracts

There are currently no implementations of contracts that meet the criteria outlined in Section 1. Even worse than that, all current implementations of contracts take a fundamentally flawed approach, because they ultimately link the transfer of coins to the decisions of the owners of private keys, instead of to external events.

3.1 Bitcoin

In Bitcoin, the closet approximation is a 2-out-of-3 multisig outpoint with a 3rd party that verifies that the transaction spending the outpoint is correct. However, this is:

- Not **reality-based**: The 3rd party approves where the coins are sent. There's no guarantee of consistency: if a particular 3rd party is currently responsible for multiple contracts that are based on the same event (say the Dow Jones' next opening price), they can resolve the contracts inconsistently, siding with one party in some cases and the other in other cases..
- Not **trustless**: The 3rd party is a single point of failure. They can even choose to refuse to sign any transaction involving the coins, thus eliminating them from the money supply.
- Not **tradable**: Once the coins are sent to the 2-out-of-3 multisig outpoint, they are locked and cannot be spent. The only way to trade the contract would be for all of the parties to agree amongst themselves that the result of one or the other side will now be sent to a 4th or a 5th party, something that requires agreements, based on trust, occurring outside of the blockchain, requiring all parties to devise, agree to and use trust-based (i.e. non-blockchain-enforceable), non-standard procedures.

3.2 Ethereum

Ethereum [10] will ultimately allow the maximum flexibility in how transactions are processed. Given they succeed in their Turing-complete implementation, any conceivable contract with any given structure could be created. Thus **tradable** contracts could be created, and the calculation of the contract structure *would* be **trustless**. However, there are two issues.

- The Turing-completeness itself leads to intractable problems. There has been extensive development on Ethereum for years yet still no end in sight. It may simply end up very expensive, possibly prohibitively expensive, to run the network.
- Even if they succeed in their implementation, the second issue is that this still wouldn't be **reality-based**. A working implementation of Ethereum would solve the simple math part of the contracts exceedingly well, but it will still do nothing to link the result of a contract to something that happens in reality. There still needs to be a way to link transfer of funds to external events.

3.3 Nxt

Nxt [15] is working on a feature called Nxt Smart Contracts [16]. As far as I could ascertain, this will be no different from Ethereum's solution.

3.4 UltraCoin

UltraCoin [18] claims to disintermediate financial markets and to allow counterpart risk free transactions. However, as it is built on Bitcoin, there is no possible way that UltraCoin can avoid using some intermediary. There is precious little on the actual implementation of UltraCoin, and all I have found are indicators that UltraCoin simply uses 2-out-of-3 multisig transactions, with Veritaseum, the company behind UltraCoin, being the 3rd party.

This is especially troubling if the company decides to take the opposing side of some of the financial transactions to help build critical mass. They would then be both the 2nd and the 3rd party in a transaction and thus have full control over where the coins go. They will effectively be able to steal all the coins in all the currently-unresolved contracts they have created. When somebody can push a button to steal money, it happens [13] every [11] time [3]. Any solutions involving multiple signatures to verify the contract are *always* subject to potential thefts because there is ultimately no way to tell whether the escrow service is also one of the two sides to the contract.

4 A Proper Implementation

I propose a three-step implementation of contracts, which at the end would result in contracts that satisfy all three necessary properties.

4.1 Step One: Subcurrencies

The first step is to implement subcurrencies, or assets, on the blockchain. The codebase must be updated to keep track not only of amounts, but which currency the amounts are in. This step solves two purposes:

1. It establishes the framework in which contracts will be **tradable**.

2. It adds the functionality of subcurrencies to the blockchain, which is a desirable feature in and of itself, as evidenced by the existence of Colored Coins [4], Mastercoin [12], and Counterparty [5]. Some of the benefits of subcurrencies are listed at [2].

4.2 Step Two: Contract Coins Resolved by Private Key Grading

The second step is to provide a mechanism for users to create and trade contracts.

4.2.1 Contract Creation

First, one party will announce to the blockchain that they have created a contract, which they will later manually assign a grade to. They prove ownership by providing a public key, while they will later sign the grading transaction with the corresponding private key. This step in and of itself does not create any new coins.

4.2.2 Engaging in Contracts with Backing Coins and Contract Coins

Next, users need a way to lock up their coins in order to engage in a contract. On a less anonymous system like Bitcoin or Nxt, this might be done using a concept of accounts. For example, I may have an account with 1000 coins in it - or an address which has the capability to spend 1000 unspent coins - and the system would keep track of the amount of coins that are currently involved in contracts for that account. If I create a contract using 200 of the coins, then the system would not let me spend any more than 800 coins until the contract is resolved.

However, this solution won't work on Cryptonote, because Cryptonote uses anonymous endpoints, not accounts or known endpoints. There is no publicly known connection between someone's public address and all the endpoints the holder of the address's private keys can spend. One potential solution would be to require users to publicly associate endpoints with their address before allowing them to use contracts. However, that would not only be inconvenient but it would greatly reduce anonymity. Instead, we need a system that works strictly with endpoints.

I have come up with a system where users spend endpoints just like any other transaction, except instead of being sent to another address to further be spent, they are converted into two new types of subcurrency called **backing coins** and **contract coins**.

Once a contract X is created, anybody in the blockchain can now mint **contract coins** of X backed by a given subcurrency Y. Y can be the main currency itself, be it Moneros or Bitcoins, or it can be a user-created subcurrency. In order to create the **contract coins**, a corresponding amount of the

currency/subcurrency coins must be spent and converted into **backing coins**. The **backing coins** represent the backing of the given **contract coins**. That is, they ensure that when the contract is resolved, whatever the grade is, the coins will be available to pay out the contract in full.

The **backing coins** represents one side of the contract, while the **contract coins** represent the other side. Since in order to mint n **contract coins**, n regular coins must be converted into **backing coins**, there are always an equal number of **backing coins** and **contract coins**.

4.2.3 Grading

Now, at a later date, the creator of the contract **grades** the contract. They assign the contract a value from 0% to 100% indicating the result of the contract. Based on the grade, the **backing coins** and the **contract coins** can be converted back into the currency they were created from. If the grade is 0%, then the **contract coins** become worthless while the backing can be unlocked into the full value. If the grade is 100%, the **contract coins** can be converted into the full value of the backing currency, while the **backing coins** are worthless. If the grade is 50%, then they each convert into half their value.

With some simple bookkeeping, it's clear that coins of the backing currency are ultimately neither created nor destroyed by this mechanism, and for any possible grade, the contract will always be able to be resolved in full for either side. Essentially, n coins of a given currency are temporarily destroyed and converted into n **backing coins** and n **contract coins**. Once the contract is graded, the n **backing coins** and n **contract coins** are converted back into a total of n currency coins.

Note that it is the contract itself that is graded, and not each holder of **contract coins** and **backing coins**. That is, all holdings of backing and contract coins for a given contract will be resolved in precisely the same way, instead of each separate holding being graded separately as they would have to be if using a 3rd-party-escrow with Bitcoin. This reflects the true nature of a contract, that it is the external result itself that matters, not each instance of making a contract based on that result.

4.2.4 Tradability

Both **backing coins** and **contract coins** are treated exactly like any other subcurrency. They can be traded, split up into different denominations, regrouped together, etc.

4.2.5 Canceling Contracts

One final note: before a contract is graded, owners of **contract coins** and their corresponding **backing coins** can agree to combine their coins and convert them back into the backing currency. This effectively cancels the contract. Note that this is only possible if all the owners of both types of coins agree both to fuse

the coins and on where the resulting currency coins are sent. This allows parties to agree to cancel a contract without waiting for it to be graded or resolved.

4.2.6 Summary

At this point, though the contracts are **tradable**, they are still neither **trustless** nor **reality-based**. However, we have moved one step closer towards **reality-based** contracts. The grader decides the result of the contract, not which coins get resolved. The difference is obvious when you consider that anybody, not just the contract creator, can mint **contract coins** based on the same contract, and that once that contract is graded, all the sides of that contract anywhere will be resolved in precisely the same way.

4.3 Step Three: Reality-Based, Trustless Contracts via Consensus

All we need to do now is to link the grade of a contract to the result of something occurring in reality, as opposed to the whims of the grader. The way to do this is to establish a consensus mechanism, with the proper incentives, to gather information from the internet and place it into the blockchain.

4.3.1 Information Specification

The first ingredient we need is a way to post a request to gather information to the blockchain. Each **info request** consists of a list of **URLs**, along with a **locator script** and a **timestamp** for each URL. The **locator script** is a non-Turing-complete script that simply points to where in the HTML/XML document the information will be, e.g. one locator might be "the bolded text, stripped of commas, of the second child of the third div". For each **data point**, all the values gotten from each URL should agree, or the requester can specify what majority is required, e.g. 2 out of 3 URLs must match, or 4 out of 5, etc.

Next, when the contract is created, the creator, instead of providing a public key which he will later use, provides a list of **data points**, along with a **mathematical formula** taking the **data point** values as inputs and returning a grade from 0% to 100%.

For example, if the contract is based on the price of the NASDAQ, there will be one **data point** representing that price along with a formula converting the price into the grade. If the contract is based on the prices of two stocks, there will two **data points**, one for each stock.

Note that the **info requests** are separate from the contracts. Thus if there are multiple contracts based on the same event, the contract creators can link to the same **data points** so that the information only has to be gathered once.

4.3.2 Information Gathering

For each **info request**, when the **timestamp** for a given URL is reached, miners can choose to start accessing URLs, extracting the data, and putting that data

into the block they are mining. While it does take longer to access a URL and extract the data than to compute a hash for a block, the miner only has to pull the data once, at which point they just include it into any block they try to hash at no additional computational or bandwidth-related cost.

To further encourage data gathering, one option is for the contract creator to charge a fee for creating the contract, taken away from each side of the contract. The creator can share part of this fee with the miners to incentivize them to gather the information. Another option is to base the amount of data the miners are willing to download on the fee that the contract creator charges.

4.3.3 Consensus

To reach consensus, the vast majority of nodes that access the websites must post information that matches, say 20 out of 25 consecutive postings of the information must match. Once consensus is reached, each node trusts that the information is valid, and evaluates the mathematical formula themselves to determine what the value of the contract is.

As a final check against the miners colluding, the contract grader posts an **approval transaction** indicating that the information gathered into the block chain is valid. If the contract grader rejects the transaction, then the consensus process starts over from scratch.

4.4 Avoiding Abuse

A few steps must be taken to avoid abuse, both on the side of the contract creators and on the side of the information gatherers.

4.4.1 Contract Creators

A contract creator might specify many URLs, or URLs that take a long time to load, or are very large, or take a long time to parse. Or he might continue to reject consensuses that are valid, to waste everyone's time. To avoid this, the network can require him to post a **bond** which only gets released once consensus is reached and he approves the contract. If he makes a difficult-to-verify contract, no miner will verify it, and the bond will never be released. Each time he rejects a consensus he must post another bond. To prevent perpetual rejections of valid consensuses, the network can force approval if a consensus reached by the miners is the same as a previous consensus.

4.4.2 Information Gatherers

As stated previously, miners who look up information should be rewarded with a percent of the contract fee. To discourage miners from just copying the previous information, miners could also be required to post a small bond when posting information. If they end up being in the minority of an accepted consensus, they lose the bond.

Thus the miners are incentivized to provide the correct information, and contract creators are incentivized to make the information-gathering process as rapid and simple as possible.

4.5 Limitations

There are a few limitations to this approach to contracts. Most notably, the full amount backing each contract must be locked from the very beginning. Thus, one couldn't really use this for insurance contracts, as the issuer of the contract would need to lock the full potential payout for each of their policies, defeating the point of insurance in the first place.

More generally, anything requiring a fractional reserve — like a bank holding fractional deposits — can't be done, since again it would require not backing contracts by the full amount. While this is a significant limitation, it makes the implementation far easier and not prone to all of the potential disasters that can beset any fractional reserve system.

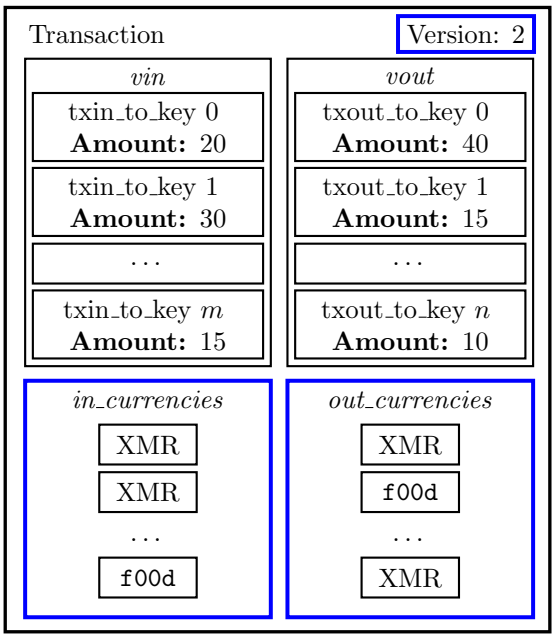
5 Implementation Details

Here's a sketch of how the steps outlined in Section 4 can be implemented on the Cryptonote code base.

5.1 Subcurrencies

5.1.1 New Currency Field

Add a new transaction version. The new transaction version contains two extra fields to keep track of which currency the inputs and outputs are in. In the old transaction version, the standard currency is assumed, e.g. XMR for Monero. Thus transactions that don't use the subcurrency feature won't take up any additional space, and transactions with the old version number will be compatible with the new blockchain.

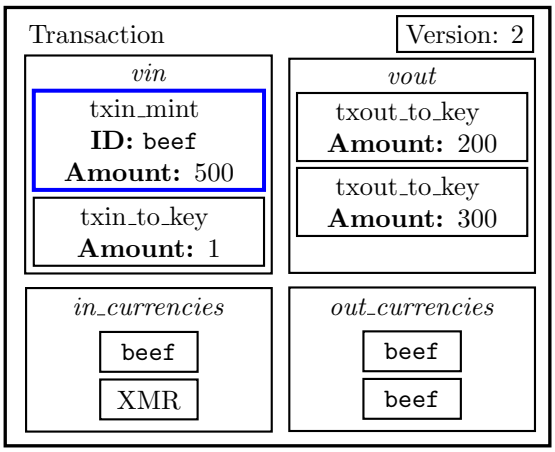


This example transaction has two inputs in XMRs, some other amount of inputs, ending with an input in a subcurrency called `f00d`. Additional bookkeeping (see 5.1.4) is required to make sure that all the outpoints used as inputs to the **txin_to_keys** are of the corresponding currency (i.e. XMR for the first two inputs, `f00d` for the last input). The outs likewise have their specified currencies.

Just like before the transaction had to be verified to make sure the outputs are not greater than the inputs, now the transaction has to be verified to ensure the outputs for each currency are not greater than the inputs for each currency. The example transaction is valid because there are 50 XMR inputs and outputs and 15 `f00d` inputs and outputs.

5.1.2 New Input Type: txin_mint

Introduce a new input type, **txin_mint**. This specifies the currency id, and the amount of coins of that currency to create. That transaction can now have outputs in the new currency up to the amount specified in the **txin_mint**. This input is only valid if it specifies a previously unused currency id, so another user can't just create more coins of an existing currency.



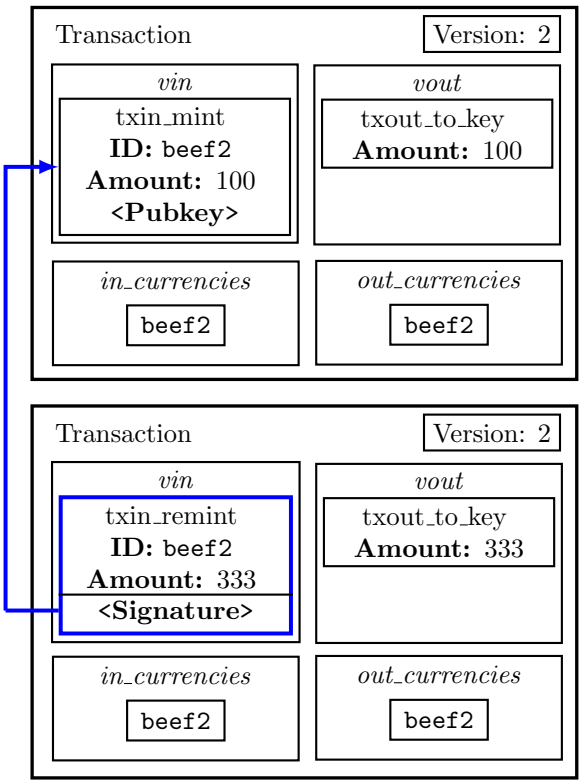
This sample transaction creates a new currency called `beef` with a money supply of 500 coins. Note that the **txin_mint** points to no preceding transaction; it is created out of thin air, as it were. However the mint is only valid if this currency has not been seen in a previous **txin_mint** input.

There's another input in XMRs to pay for the transaction fee. The resulting **beef** coins are sent to two different destinations. Note that these are standard **txout_to_keys** that can be spent by standard **txin_to_keys** in precisely the same way as coins of the standard currency.

5.1.3 New Input Type: txin_remint

A user might want to be able to create more coins of the currency he created later on. For this purpose, he can specify a public/private key pair when he first creates his currency. Then he can create a transaction with another new input type, a **txin_remint**, specifying which currency to create more coins of and how many. For the **txin_remint** to be valid, it must be signed by the key pair in the initial **txin_mint**.

Note that the public key he provides can be the same as his public address, or it can be completely different. Thus the currency minter can choose to be anonymous or publicly known.



The transaction above demonstrates a successful remint transaction. The **txin_remint** includes a signature signing the input which is checked using the public key that was provided in the **txin_mint** input of the currency of the same ID. The remint is successful if and only if the currency has already been created and the signature is valid for the public key already in the blockchain. The remint transaction can also include (not shown here) a new public key to use for reminting after this transaction. Thus a currency owner can pass minting capabilities to somebody else without having to exchange any private keys. He can also set the remint key to null, indicating that no new units of the currency will ever be created.

5.1.4 Updated Bookkeeping

Then, all that needs to be done is to update the logic of the blockchain implementation so that the outputs are indexed not just by the amount of the outpoint, but by the currency as well. That is, modify the following line in `cryptonote_core/blockchain_storage.h`:

```
typedef std::map<uint64_t,
                 std::vector<std::pair<crypto::hash, size_t>>>
        outputs_container;
```

To:

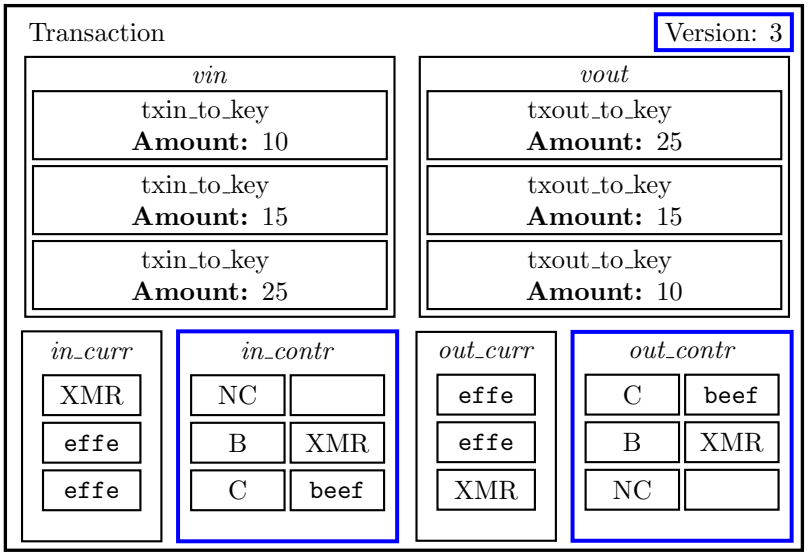
```
// the currency/subcurrency of the coin
typedef uint64_t coin_type;
// the key is now the type of coin and the amount
typedef std::pair<coin_type, uint64_t> output_key;
typedef std::map<output_key,
                 std::vector<std::pair<crypto::hash, size_t>>>
        outputs_container;
```

Then, update all the code that indexes and adds to it to differentiate outpoints by the new key. The result is to neatly separate all coins of the different currencies so that they can't be mixed, and that ring signatures allowing certain coins to be spent won't accidentally validate spending coins of other subcurrencies.

The standard currency and the subcurrencies will be treated precisely the same way. The cryptographic innovations of Cryptonote will apply to subcurrency transactions as well; they, too, can be sent unlinkably and untraceably.

5.2 Contracts

5.2.1 New Contract Type and Backing Currency Fields



Similar to how we added a currency field to every transaction, we now increase the version number and add two more optional fields to keep track of the contract information for the inputs and outputs. The contract information consists of a pair: the **contract type** and the **backing currency**. The **contract type** distinguishes between three possibilities: **no contract** (NC in the diagram), a **backing coin** (B), or a **contract coin** (C).

All previous transactions, using either the standard currency or the user-created currencies, are categorized as **no contract**. For **backing/contract coins**, the currency field is used to indicate which contract the coins are for, the **contract type** is used to distinguish between whether they are **backing coins** or **contract coins**, and the **backing currency** is used to indicate which currency the contract is being backed by, that is, which currency the coins will be converted into once the coin is graded.

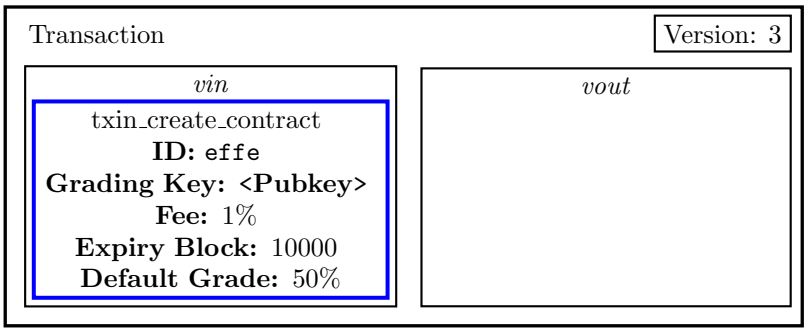
The sample valid transaction demonstrates the new fields. There are three inputs: one standard input of XMRs, one backing-coins input of a contract called `effe` backed by XMRs, and one contract-coins input of that same contract, yet backed by the `beef` subcurrency. Just like we updated the bookkeeping to keep track of currencies, we must now also update the bookkeeping to keep track of contract types and backing currencies. The coin's type, instead of being uniquely identified by the currency id, is now uniquely identified by the three-tuple `<currency, contract_type, backing_currency>`.

The sample transaction is valid because none of the outputs spend more than any of the inputs.

5.2.2 New Input Type: txin_create_contract

This input type announces to the blockchain that a new contract has been created. It provides the public key that will be used to grade the contract. Here a fee could also be specified. To avoid contracts that never resolve, the creator must give an expiry block along with the default grade should the contract expire.

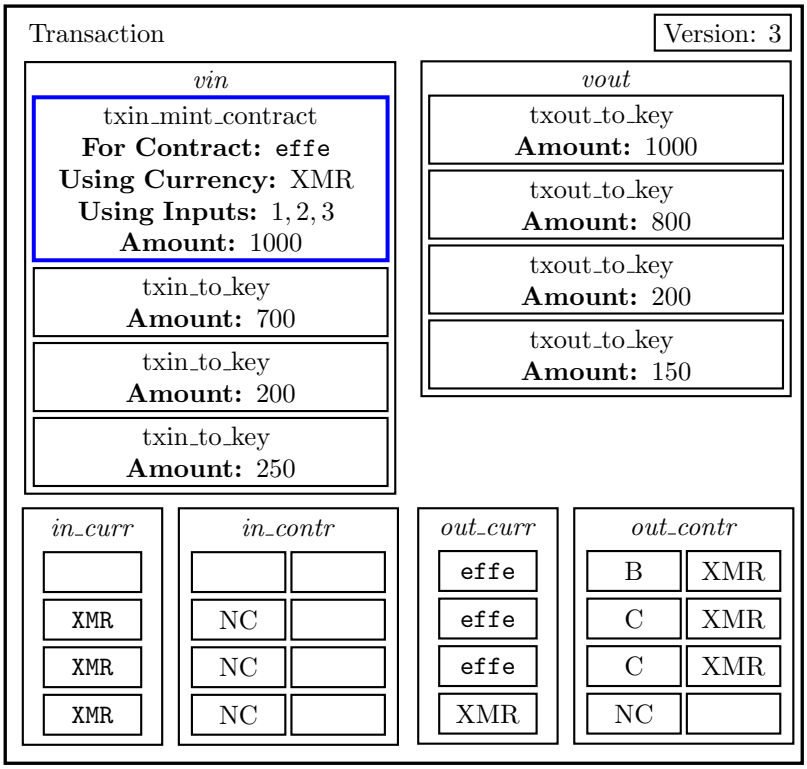
This input creates no new coins of any currency.



This transaction creates the contract called `effe`, provides the key that will be used to check the grading transaction signature, and indicates the default grave should the contract expire — that is, should there be no grading transaction by block 10000. Note that as this input type creates no coins and no outputs, there is no particular need to have any outputs or currency or contract type fields.

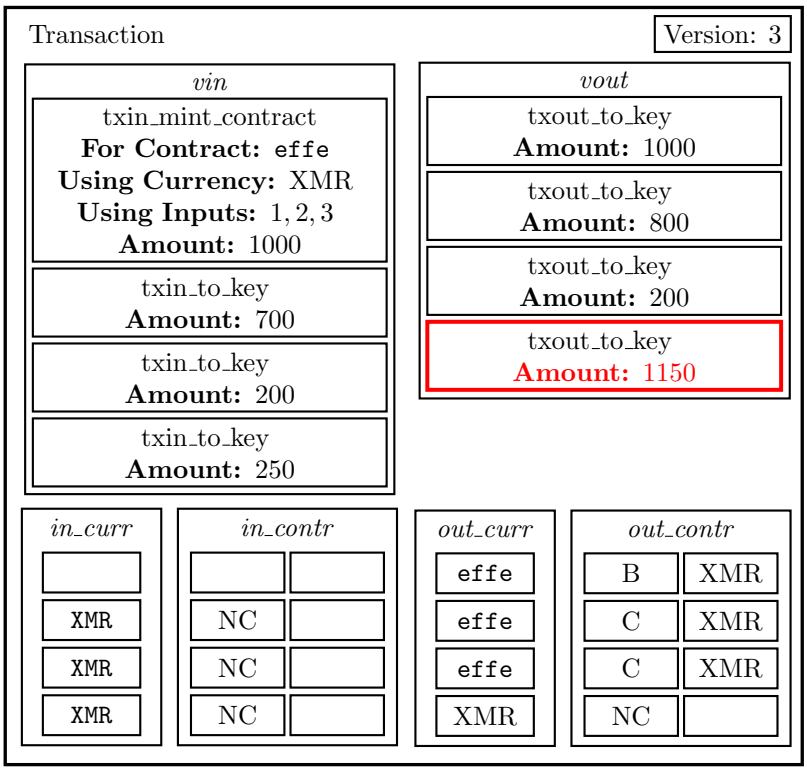
5.2.3 New Input Type: txin_mint_contract

This input type converts currency or subcurrency coins into contract and backing coins backed by said currency or subcurrency. It indexes into the other transaction inputs and then allows the spending of n **contract coins** and n **backing coins**.



This transaction demonstrates minting 1000 contract and backing coins of the `effe` contract using 1150 XMRs, sending 150 XMRs back as change (the 4th output). The **txin_to_keys** are processed as usual. The mint contract indexes into the inputs to indicate which inputs to use to mint the transaction.

If successful, the mint input allows the spending of 1000 **backing coins** and 1000 **contract coins**, with the backing currency specified in the mint contract (XMR in this case). It does *not* allow also spending those 1000 XMRs in other **txout_to_keys**. That is, the following would be an *invalid* transaction:

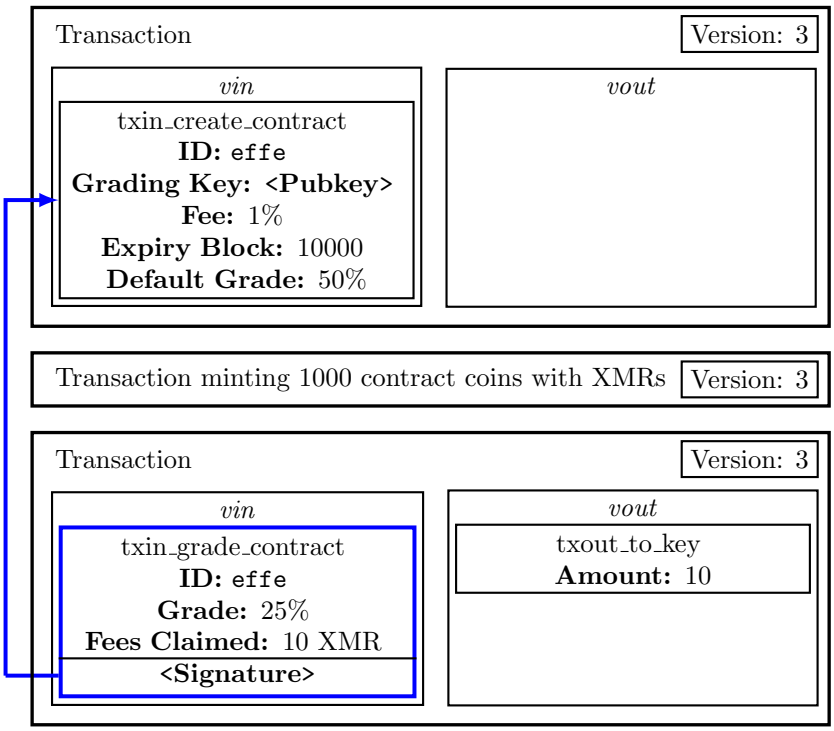


This is invalid because, although there were 1150 XMR inputs and 1150 XMR outputs, the **txin_mint_contract** consumed 1000 of those XMRs to create the **backing** and **contract coins**.

Note how all of these transactions still use the anonymity features that Cryptonote provides. If the **txin_to_keys** are mixed, nobody can determine which outputs were spent to mint these coins. Further, nobody can link the outputs that the coins are sent to to any particular address, unless the owner of that address chooses to reveal it.

5.2.4 New Input Type: txin_grade_contract

This input type grades a given contract. It provides the grade itself, and signs the grade with the private key corresponding to the public key provided in the **txin_create_contract**. This transaction is the one which allows the creator to spend the fees, if any. For example, if the fee charged is 1%, and 1000 **contract/backing coins** had been created using XMRs, then this input allows spending 10 XMRs.

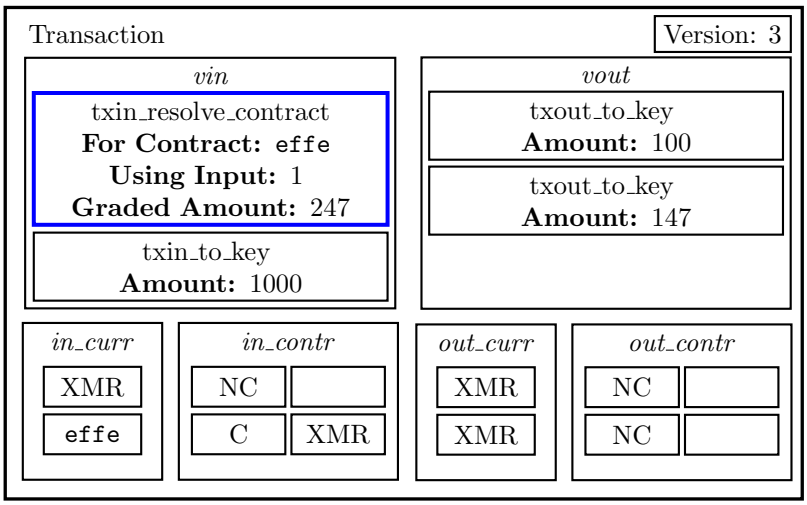


This transaction grades the contract called `effe` at 25%, meaning the holders of **contract coins** can convert them back to 24.75% of the value (25% minus the 1% fee) and the holders of **backing coins** can convert them into 74.25% of the value (75% minus the 1% fee). The transaction also spends 10 XMRs as fees — that is, 1% of the 1000 XMRs that were used to mint the contract/backing coins.

In order to be able to validate this contract, nodes must keep track of how many coins used a given contract, so that the fee can be calculated and validated.

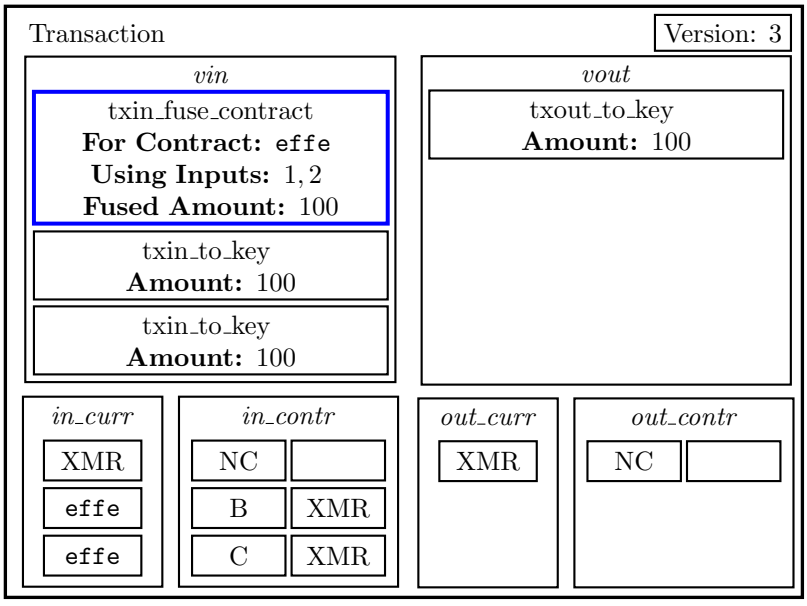
5.2.5 New Input Type: txin_resolve_contract

This input type contains a **txin_to_key** which spends a **backing coin** or a **contract coin** and allows spending the appropriate amount of coins of the the currency that they were created from. The amount that the backings and contracts can be converted to depends on the grade and the fee, as specified in the previous section.



This transaction demonstrates resolving 1000 **contract coins** of contract `effe` (from the 2nd input) into the graded amount of 247 ($1000 * 0.25 * 0.99$, rounded down). A similar transaction resolving 1000 **backing coins** would be able to spend 742, instead ($1000 * 0.75 * 0.99$, rounded down). All amounts are always rounded down to prevent coins from being created via rounding errors.

5.2.6 New Input Type: txin_fuse_contract



This input type contains an index into other **txin_to_keys** along with an amount to fuse. The referenced **txin_to_keys** should spend at least the given amount of **contract coins** and **backing coins** of the same contract and cur-

rency. If successful, the input allows spending the fused amount of coins of the underlying currency, as well as spending the remainder of the un-fused **contract coins** and **backing coins**.

The sample transaction fuses 100 backing coins with 100 contract coins, both for the `effe` contract using XMR as a backing currency, back into XMRs. Note that in order to prevent avoiding paying the contract grading fees, once a contract is graded, fusing is no longer allowed.

5.2.7 Updated Bookkeeping

Once more, the bookkeeping must be updated. Now, instead of outpoints being separated by currency and amount, they must be separated by currency, contract type, backing currency, and amount, as described in the Section 5.2.1. Again, so long as all of these fields match, transactions can mix and match different outputs so that they can be sent untraceably and unlinkably. All that must be done is to update what `coin_type` was from the previous code listing:

```
//typedef uint64_t coin_type;
enum CoinContractType {
    NotContract = 0,
    ContractCoin = 1,
    BackingCoin = 2
};
// <currency, contract_type, backing_currency>
typedef std::tuple<uint64_t, CoinContractType, uint64_t> coin_type;
```

5.3 Consensus

Now for the most important part, the consensus information gathering.

5.3.1 New Input Type: txin_info_request

This input type contains the information outlined in Section 4.3.1:

```
struct txin_info_request
{
    uint64_t data_point_id;
    string URL;
    locator_script locator;
    uint64_t start_time;
};
```

The **data_point_id** uniquely identified the data point. The rest provide the specification of where to get the information from.

5.3.2 Modify txin_create_contract

Allow the option still to provide the grading key, but provide another option involving data points:

```
vector<uint64_t> data_point_ids;
formula grade_calculation;
```

The **data_point_ids** reference previously-created info requests. I haven't fully specified the format of `formula` yet, but I envision it would be a tree where the leaves are constants or variable references, and the nodes are simple operations such as addition, multiplication, substring operations, converting strings to integers, etc.

5.3.3 New Coin Type: contract bond coin

The simplest way to implement bonds is to introduce a new coin type, a **contract bond coin**. This can be done by extending the `CoinContractType` defined in Section 5.2.7 with a `ContractBond` type. In order for a **txin_create_contract** using the information-gathering option to be valid, it must refer to a valid **txin_to_key** in the current transaction. The input is spent and converted into a **contract bond coin**, which can only be converted back into XMRs with a new input type. As a side benefit, the **contract bond coins** themselves can be traded just like any other coin.

5.3.4 New Input Type: txin_release_bond

A **txin_release_bond** input takes **contract bond coins** as input and converts them back into the underlying standard currency — XMRs, in our running example. The input is only valid if the contract has already been graded via consensus. If the contract expires, then the **contract bond coins** for that contract can never be released.

5.3.5 Modify Block to Allow Data Points

Augment the block structure to allow for providing information, adding the following after **tx_hashes**:

```
struct data_point {
    uint64_t data_point_id;
    uint64_t time_retrieved;
    size_t consensus_round;
    string data_result;
};
vector<data_point> data_points;
```

The **data_point_id** identifies the **data point** which information is being gathered for. **data_result** is the actual data retrieved after evaluating the

script. The **consensus_round** indicates which round the data is being gathered for. This is important when resolving miner bonds (see next section).

For compatibility, the **data_points** are optionally serialized depending on the block version. Thus old blocks will still be compatible after the fork.

5.3.6 Miner Bonds

5.3.6.1 New Coin Type: miner bond coin Analogous to the contract creator bond coin, to implement miner bonds we introduce another coin type, the **miner bond coin**. **Miner bond coins** will be uniquely identifiable by the contract, variable, data result, and consensus round they are posted for.

5.3.6.2 New Input Type: txin_release_miner_bond This input type will take an input of a **miner bond coin** and convert it back into the main currency. It will only be valid if that consensus round was cancelled by the contract creator, or if the consensus that was reached contains the same data result value for the given variable.

5.3.6.3 New Input Type: txin_post_miner_bond This input type takes a **txin_to_key**, along with an index into the block's **data_points**, spends the input, and creates the appropriate **miner bond coin**. It is only allowable in a special transaction in the block, the **bond_tx**.

5.3.6.4 bond_tx For miners to post the bonds, add to the end of the block structure a new field called **bond_tx**. The **bond_tx** must contain one post miner bond input per **data_point**, with enough money posted to cover whatever the bond cost is. For space efficiency it may be better to hard code these especially, to avoid the overhead of using a generic transaction. Ultimately you only need a vector of pairs of **txin_to_key**s hard-coded to spend XMRs and **txout_to_keys** hard-coded to send miner bond coins with the parameters drawn from the corresponding **data_point**.

5.3.7 New Input Type: txin_approve_reject_information

This input type contains a **contract_id** and indicates whether the contract owner approves or rejects the consensus determined by the miners. It is only valid if a consensus has been reached. It is signed with the private key corresponding to the public key provided in the **txin_create_contract**. It also contains the grade for easy reference, calculable by any node from the information already placed into the blockchain.

If the consensus is approved, then the contract is treated as if it was graded directly without consensus gathering, **contract coins** and **backing coins** can both be turned back into their respective currencies, etc.

As described earlier, the consensus is only rejectable if a previous consensus with the same result has not already been reached.

6 Conclusion

This is the way forward for implementing contracts. The first cryptocurrency that successfully implements the above will go above and beyond the others in terms of features. This is all relatively easy to implement and doesn't require any original research or solving any intractable problems, like attempting to create a viable Turing-complete language that a decentralized, distributed network can execute.

I am willing to do the work it takes to implement the above on any given Cryptonote-based coin, be it ByteCoin, Monero, Boolberry, Dashcoin, etc. I am willing to be paid in the currency itself since I believe this will greatly increase its value. Please contact me at the email at the start of this document to discuss further logistics, payment details, etc.

References

[1] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System.* https://bitcoin.org/bitcoin.pdf.

[2] Meni Rosenfeld, *Overview of Colored Coins.* https://bitcoil.co.il/BitcoinX.pdf. December 4, 2012. Mastercoin [12], and Counterparty [5].

[3] Patrick McGuire, *Over $100K in Bitcoin Was Stolen in a Ridiculously Low-Tech Heist.* https://news.vice.com/article/over-100k-in-bitcoin-was-stolen-in-a-ridiculously-low-tech-heist. March 19, 2014.

[4] *Colored Coins.* http://coloredcoins.org/

[5] *Counterparty.* http://counterparty.io/

[6] Nicolas van Saberhagen, *CryptoNote v 2.0.* https://cryptonote.org/whitepaper.pdf. October 17, 2013.

[7] Evan Duffield, Kyle Hagan, *Darkcoin: Peer-to-Peer Crypto-Currency with Anonymous Blockchain Transactions and an Improved Proof-of-Work System.* https://www.darkcoin.io/wp-content/uploads/2014/09/DarkcoinWhitepaper.pdf. March 18, 2014.

[8] https://www.darkwallet.is/

[9] gmaxwell, *CoinJoin: Bitcoin privacy for the real world.* https://bitcointalk.org/index.php?topic=279249.0. August 22, 2013.

[10] https://www.ethereum.org/

[11] Andy Greenberg, *Another Bitcoin Startup Tanks After $600,000 Theft.* http://www.forbes.com/sites/andygreenberg/2014/03/04/another-bitcoin-startup-tanks-after-600000-theft/. March 4, 2014.

[12] *Master Protocol & Mastercoin*. http://www.mastercoin.org/

[13] Robert McMillan, *The Inside Story of Mt. Gox, Bitcoin's $460 Million Disaster*. http://www.wired.com/2014/03/bitcoin-exchange/. March 3, 2014.

[14] Surae Noether, *Review of Cryptonote White Paper*. http://monero.cc/downloads/whitepaper_review.pdf. July 14, 2014.

[15] http://nxt.org/

[16] http://www.nxttechnologytree.com/nxt-technology/nxt-smart-contracts

[17] *Shared Coin — Free Trustless Private Bitcoin Transactions*. https://sharedcoin.com/.

[18] Reggie Middleton, http://ultra-coin.com/