BitcoinX Colored bitcoins (Coloredcoins.org)

Yoni Assia yoni.assia@etoro.com - Vitalik Buterin v@buterin.com
Leor Hakim liorhakim@gmail.com - Meni Rosenfeld meni@bitcoil.co.il

Abstract

Bitcoin is the world's first decentralized digital currency, allowing the easy storage and transfer of cryptographic tokens, using a peer-to-peer network to carry information, hashing as a synchronization signal to prevent double-spending, and a powerful scripting system to determine ownership of the tokens. There is a growing technology and business infrastructure supporting it.

By the original design bitcoins are fungible, acting as a neutral medium of exchange. However, by carefully tracking the origin of a given bitcoin, it is possible to "color" a set of coins to distinguish it from the rest. These coins can then have special properties supported by either an issuing agent or by public agreement, and have value independent of the face value of the underlying bitcoins. Such colored bitcoins can be used for alternative currencies, commodity certificates, smart property, and other financial instruments such as stocks and bonds.

Because colored bitcoins make use of the existing Bitcoin infrastructure and can be stored and transferred without the need for a third party, and even be exchanged for one another in an atomic transaction, they can open the way for the decentralized exchange of things that are not possible by traditional methods. In this paper we will discuss the implementation details of colored bitcoins and some of their use cases.

Background

The advent of Satoshi Nakamoto's Bitcoin in 2009 has fundamentally enlarged the scope of what can be done with cryptography. Before Bitcoin, any attempt to manage any kind of digital assets online without a centralized authority always ran into a fundamental problem that was thought to be intractable: the double-spending threat. If a digital asset is nothing but a file consisting of ones and zeroes, the theory went, what stops anyone from simply making a hundred copies of that file and thereby multiplying their own wealth? With a central authority, the problem is easy; the central authority can simply keep track of who owns what quantity of the digital asset. However, centralization is problematic for many reasons: a centralized authority creates a single point of failure which can be hacked, shut down due to regulatory pressure, or even leverage its privileged status to benefit itself at the wider community's expense. Although some

abortive attempts were made by Wei Dai and Nick Szabo in 1998 and 2005, Bitcoin marks the first time that anyone has created a fully functional, viable decentralized solution - a solution which is rapidly gaining popularity and mainstream acceptance all around the world.

Because of its fully digital and decentralized design, Bitcoin offers a number of advantages over existing payment systems. Its transactional irreversibility provides for a high degree of security, allowing merchants to considerably cut costs in high-risk industries. Its lack of a centralized authority has rendered it invulnerable to so-called "financial censorship" attempts such as the financial blockade of Wikileaks by banks and credit card companies in 2011; while donations through the traditional financial system dropped by 95% as a result of the blockade, Bitcoin donations kept coming in at full steam the whole way through[1]. Its technical ease of use is allowing entrepreneurs around the world to start up gambling sites, prediction markets, paywalls and many other types of businesses with zero startup cost in only a few weeks' worth of development work. Finally, most interestingly of all, the public nature of Bitcoin transactions potentially allows for a degree of radical transparency for businesses and nonprofit organizations that is simply not possible any other way; although this area has not yet been fully developed in practice, one early example of such a system being implemented in practice is the "provably fair gambling" systems used by sites like SatoshiDice, allowing users to cryptographically verify that the site is not cheating on bets. Given all of these advantages, the natural question is: is it possible to use the same functionality for other applications as well?

The answer, as it turns out, is yes. The fundamental innovation behind Bitcoin, that of using cryptographic proof of work to maintain a secure distributed database, is good for more than just the single limited-supply currency originally envisioned by Satoshi Nakamoto in 2008; exactly the same technology can be used to maintain ownership of company shares, "smart property", alternative currencies, bank deposits and much more. Anything which is (1) a digital asset, and (2) a "rivalrous good", meaning that only one person can own it at a time, is potentially fair game for representation in the Bitcoin blockchain. However, Bitcoin does not include any facilities for doing this by default; to do any of these things, an additional protocol is required. And BitcoinX intends to accomplish just that.

Overview

The idea behind colored coins is to separate the bitcoin network and its underlying cryptographic technology from the bitcoin as a currency unit and the surrounding ecosystem, using the Bitcoin network simply as a mechanism to clear transactions and avoid double spending.

There are many potential use cases:

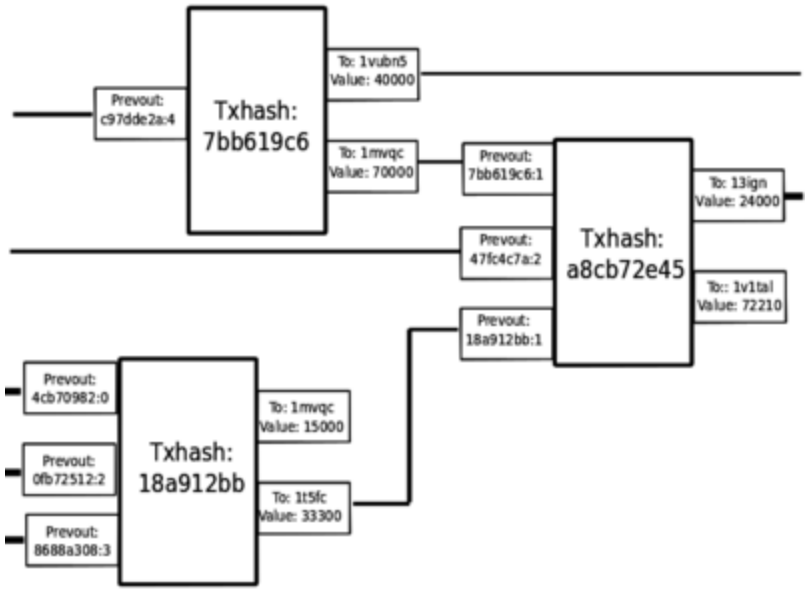
- A company might want to **issue shares** using colored coins, leveraging the Bitcoin infrastructure to allow people to maintain ownership of shares and trade shares, and even allow voting and pay dividends over the Bitcoin blockchain.

- **Smart property**: suppose there is a car rental company. The company can release one colored coin to represent each car, and then configure the car to turn on only if it receives a message signed with the private key that currently owns the colored coin. It can then release a smartphone app that anyone can use to broadcast a message signed with their private key, and put up the colored coins on a trading platform. Anyone will be able to then purchase a colored coin, use the car for whatever period of time using the smartphone app as a "car key", and sell the coin again at their leisure.
- A local community might wish to create a **community currency**, using the Bitcoin infrastructure to securely store funds.
- A company may wish to create a **corporate currency**, such as Air Miles rewards points, or even plain **coupons**.
- A bank might wish to release a coin to **represent deposits**, allowing people to trade, for example, "USD coins" or "gold coins".
- Decentrally managing ownership of **digital collectibles** such as original artworks - just like art collectors buy and sell original copies of famous paintings for millions of dollars today, colored coins allow us to do the same with purely digital objects, such as songs, movies, e-books and software, as well, by storing the current ownership of the work as a colored coin on the blockchain.

Technical background: Bitcoin transactions

The fundamental object in the Bitcoin network is called a Bitcoin transaction. A Bitcoin transaction contains:

1. A set of **inputs**, each of which contains (i) the **transaction hash** and **output index** of one of the outputs of a previous transaction, and (ii) a **digital signature** which serves as cryptographic proof that the recipient of the previous transaction output authorizes the transaction.
2. A set of **outputs**, each of which contains (i) the **value** (amount of BTC) going to the output, and (ii) a **script**, which determines the conditions under which the output can be spent.



The precise nature of scripts and signatures is not important for the purposes of the colored coins protocol; a useful simplification is to think of the output script as a public key and the signature as a standard cryptographic digital signature made with the corresponding private key. Every output script maps to a unique Bitcoin address, and it is possible to efficiently convert back and forth between the two representations. For example, an output script looks like this:

`76a914845c67f3c353761af089c27fec531f6138d5a62f88ac`

And its corresponding address looks like this:

`1D4rtoqGJqEgB8JmwkoBboxA9tmRELfMCy`

When you are sending money to a Bitcoin address, your Bitcoin client is actually making a Bitcoin transaction with one of its outputs having the output script corresponding to that address.

Transactions also have a special property known as lock time, and each input has a signature type and sequence number, but these features are not essential to understanding how Bitcoin works and will be introduced as needed.

The algorithm for validating a transaction is as follows:

INPUT_SUM = 0

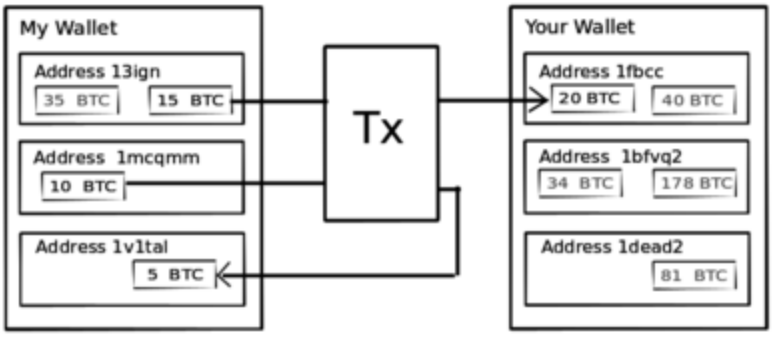
```
for input in inputs:
    prevtx = get_tx_from_blockchain(input.prev_out.hash)
    if prevtx == NULL:
        return FALSE
    if is_output_already_spent(prevtx,input.prev_out.index):
        return FALSE
    if not verify(input.sig,
```

```
    prevtx.outputs[input.prev_out.index].script):
            return FALSE
        INPUT_SUM += prevtx.outputs[input.prev_out.index].value
    OUTPUT_SUM = 0
    for output in outputs:
        OUTPUT_SUM += output.value
    if INPUT_SUM < OUTPUT_SUM:
        return FALSE
    return TRUE
```

Note that Bitcoin does not recognize the concept of "address balances"; that is simply a layer of convenience added on top by Bitcoin software. If your Bitcoin wallet tells you that you have 50 BTC, what that means is that there are 50 BTC worth of transaction outputs (perhaps one output of 50 BTC, perhaps four of 12.5 BTC; any combination works) whose output scripts correspond to your address.

But what if there is no set of transaction outputs that you can combine into a transaction to send exactly what you want? In that case, Bitcoin uses a concept called change. Essentially, you put more than you need to into the transaction, but you send the excess back to yourself in one of the outputs.

Usually, change goes into a new address to preserve privacy. Using these transactions, it's possible to maintain user account balances and send to other users any amount desired.



Genesis transactions

[TODO LATER]

Transfer transactions

The more challenging part of developing the colored coins protocol is to determine a way of transferring colored coins from one address to another. For transactions with one input and one output, the problem is trivial; the color of the output can simply be the same as

the color of the previous output that is spent by the input. If a transaction has multiple inputs and outputs, however, the situation is different; how do we know which inputs correspond to which outputs? As it turns out, figuring out a way of specifying this is more challenging than it seems; although several solutions do present themselves, they are all imperfect in their own ways.

Formally, what we care about is defining a function, ComputeOutputColors, that takes a transaction containing inputs and outputs, of the form:

```
inputs = [
       {
           color: [col],
           value: [integer],
           colorvalue: [integer],
       },
       ...
   ]
   outputs = [
       { value: [integer] },
        ...
   ]
```

And computes the "color" and "colorvalue" property of each output. In simpler coloring schemes, we have value = colorvalue, but this does not always hold true for more complex schemes. The difference between the two is this: an output's value is the number of satoshis in the output, according to the plain Bitcoin protocol, whereas its "colorvalue" is the number of colored satoshis that it has. For example, in padded order based coloring (see below) a transaction may have 60000 regular satoshis, but only 50000 colored satoshis. It could also go the other way around; under certain tagging-based coloring schemes, an output might have 10000 satoshis according to the Bitcoin procotol, but 10 million satoshis according to the coloring rules.

All coloring schemes described in this section, with the exception of per-satoshi tracking, have the following similarities:

- Every output has a color. We define the "uncolored" color to represent uncolored transaction. By convention, any output of the "uncolored" color has a colorvalue of zero.
- Every output has one or more "parent" inputs. The color of an output is either (i) the color of the parent inputs, if all parent inputs have the same color, or (ii) "uncolored" if any two parents have different colors.

Note that this conflicts somewhat with the above definition of a coloring scheme, where it is introduced as a function F(output,color) -> colorvalue - here, we do assume the existence of multiple colors using the same coloring scheme, and we determine both the color and the colorvalue of each output. To show that the two approaches of thinking about color are equivalent within the context of a single coloring algorithm, we prove a theorem:

Coloring Independence Theorem: Evaluating a coloring algorithm over multiple colors { U,

C_1 ,C_2 ... C_n} (with U = uncolored) and evaluating the coloring algorithm independently over { U, C_i }for all i will always produce the same result for any transaction output.

This theorem has three implications:

1. The question of whether or not a transaction has color X does not at all depend on the existence, nonexistence or distribution of any color other than X itself.
2. If a client knows about 1000 colors, it does not need to repeat the traversal algorithm 1000 times, once for each color. It can process every color simultaneously at minimal cost (provided that the algorithm is the same).
3. If a colored coin client that already knows about colors { C_1 ... C_n } suddenly learns about a new color, it does not need to re-evaluate the transactions associated with C_1 ... C_n; it simply needs to rerun the algorithm for the new color.

Proof: For each output, there are three cases:

1. All parents of the output have color C_i for some i. In this case, the coloring algorithm over { U, C_1, C_2 ... C_n } will give C_i as the color of the output, the coloring algorithm over { U, C_i }will give C_i as the color of the output (as all parents have color C_i), and the coloring algorithm for { U, C_j } for j != i will give U as the color of the output (as all parents do not have color C_j and so are assumed to have color U). Hence, both algorithms will report that the output has color C_i and only C_i.
2. All parents of the output have color U. In that case, all coloring algorithms will return U.
3. The parents of the output have multiple colors, C_i[1] ,C_i[2] ... C_i[m] and possibly U. In this case, the coloring algorithm over { U, C_1 ,C_2 ... C_n } will see outputs from multiple colors, and so will assign the output the color U due to the mixed coloring rule. The coloring algorithm over any { U, C_i[j] } for any j will see parent inputs having color C_i[j], but it will also see those parent inputs having some color C != C_i[j] as having color U, so it will default to U due to the mixed coloring rule.

[TODO: update the proof to handle genesis transactions]

A full colored coins implementation, including the genesis transaction protocol and the transfer transaction protocol, should ideally have the following properties:

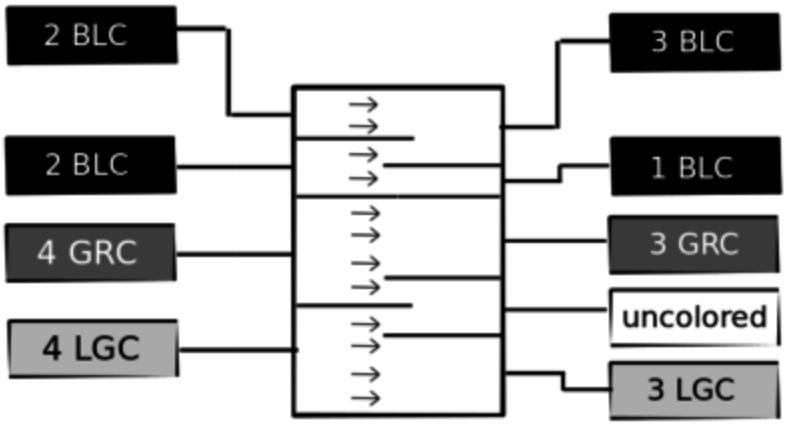
- **Simplicity** - the protocol should be as simple and easy to implement as possible. Ideally, a human should be able to look through an ordinary blockchain explorer and trace colored coins of a specific color down the transaction graph.
- **Non-restrictiveness** - as much of what one can do with bitcoins as possible should also be doable with colored coins. The transfer transaction protocol should not impose so many restrictions that it becomes impossible to, for example, make a multisignature transaction, a partially signed transaction or a time locked transaction. Unfortunately, 100% non-restrictiveness is impossible given the current protocol; for example, a trust-free assurance contract, done by publishing a transaction with a given output and allowing anyone to add and sign their own input using SIGHASH_ANYONECANPAY to pool together funds to pay for the output, cannot work, as there is no way to distinguish a colored output from an uncolored output in scripts. However, the protocol should leave as much room as possible.
- **Atomic tradeability** - it should be possible to create a single transaction sending X units

of color C_1 from A to B and Y units of color C_2 from B to A for any C_1 and C_2 provided that either (i) C_1 andC_2 use the same color protocol, or (ii) C_2 = uncolored bitcoins.

- **Efficiency** - it should be possible to efficiently compute F(output,color) -> colorvalue, ideally without preprocessing the entire blockchain. Efficiency in this context does not have a clear definition; it is simply a question of practicality.
- **Space efficiency** - colored coins transactions should impose a minimal amount of extra bloat on the Bitcoin blockchain compared to regular transactions.

Order-based Coloring

Order-based coloring is the original, and simplest, coloring algorithm. The best intuitive explanation for order-based coloring is this: say that every transaction has a "width" proportional to its total input value. One side of the transaction will have inputs, with each input having a width proportional to its value, and the other side will have outputs, with each output having a width proportional to its value. Now, suppose that colored water is flowing down (or right) in a straight line from the inputs to the outputs. The color of an output will be the color of the water that flows into the output, or uncolored if an output receives water from multiple colors.



Formally, the algorithm is defined as follows:

```
i = 0
offset = 0
for j in [0 ... length(outputs) - 1]:
    col = inputs[i].color
    while offset < outputs[j].value:
        offset += inputs[i].value
        i += 1
        if inputs[i].color != col: col = "uncolored"
    outputs[j].color = col
    if col != "uncolored":
        outputs[j].colorvalue = outputs[j].value
```

```
        offset -= outputs[j].value
    return outputs
```

Evaluation of OBC:

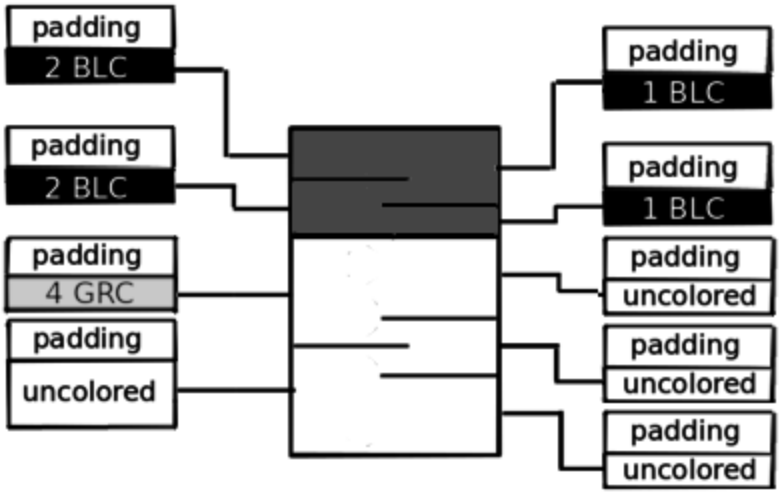
- **Simplicity**: the algorithm is easy to code, and follows an intuitive model of the way that coloring "should" work.
- **Non-restrictiveness**: trustless assurance contracts (and anything using SIGHASH_ANYONECANPAY) are impossible since scripts cannot distinguish between colored and uncolored coins. Aside from that, there are no restrictions.
- **Atomic tradeability**: atomic trades are very easy: input 0 and output 1 belong to A, and input 1 and output 0 belong to B.
- **Efficiency**: efficiency is very problematic. There are two ways to calculate the color of an output: top-down, calculating the color of every transaction output starting from the genesis output by recursively applying the OBC algorithm, and bottom-up, starting from the output and looking back through as few ancestors as possible to conclusively determine the color. The top-down algorithm is obviously not efficient without precomputation. The bottom-up algorithm is also not efficient, as every output can have multiple parents, in some circumstances causing the number of ancestors that needs to be searched to blow up exponentially. Uncolored outputs are especially difficult to prove, as the entire blockchain would potentially need to be scanned all the way back to the coinbase transactions.
- **Space efficiency**: OBC transactions take up exactly as much space as regular Bitcoin transactions.

Order-based coloring, however, also has another weakness: the dust limit. Bitcoin enforces the requirement that all outputs in a standard Bitcoin transaction must contain at least 5430 satoshis (0.0000543 BTC), and transactions that fall afoul of this requirement will take days to confirm. This introduces the following problems:

- Unique or near-unique colored coins (eg. smart property, original art, digital baseball cards,etc) need to be treated specially, since it is very difficult to send an output of exactly one satoshi. The likely solution is to create a 6000-satoshi genesis output, and then treat the 6000 satoshis together as a single colored coin; anything less than 6000 in a single output would count as nothing. However, there is a choice to be made here: can the 6000 satoshis be split and then put back together again? If so, then there is no final criterion for coin destruction, which is potentially very problematic. If not, then the implementation does not constitute pure order-based coloring.
- Stocks or currencies issued on top of colored coins would either have high minimum transaction values, or be expensive to set up. For example, if a company makes an IPO worth 10000 BTC on top of 100 BTC, the minimum amount of the stock that could be transacted would be worth 0.00543 BTC (~$1 as of the time of this writing). Note that on a lower level this is a problem for all coloring schemes described here; it is very expensive to create a color with more granularity than a few million units. Theoretically, however, it is possible to create coloring schemes that artificially addmore granularity if the need arises.

Padded Order-Based Coloring

Padded order-based coloring is a somewhat more complicated approach that addresses some of the concerns around order-based coloring. Essentially, the idea is to treat every transaction output as having a "padding" of a certain number of uncolored satoshis, with the colored satoshis following. Aside from this, the algorithm is similar to OBC, although it is more restrictive.



The algorithm works by splitting up the list of transaction inputs and outputs into "sequences", where a sequence consists of a set of inputs and outputs that perfectly line up with each other. For example, if a transaction has inputs of size 6, 6, 6, 7, 5, 3 (not including padding) and outputs of size 9, 9, 10, 2, 2, 1, then the sequences would be [[6,6,6],[9,9]], [[7,5],[10,2]] and [[3],[2,1]]. Within each sequence, if all inputs have the same color then the outputs are of that color, and otherwise the outputs are uncolored. This extra layer of restrictiveness is added to allow clients to recognize a large subset of normal Bitcoin transactions as uncolored and therefore immediately discard them. The code is as follows:

```
j = 0
offset = 0
padding = 10000

sequences = []
current_input_seq = []
current_output_seq = []

for i in [0 ... length(inputs) - 1]:
    if inputs[i].value < padding:
        break
    offset += (inputs[i].value - padding)
```

```
        while offset > 0:
            if outputs[j].value < padding:
                break
            offset -= (outputs[j].value - padding)
            current_output_seq.append(outputs[j])
            j += 1
        if outputs[j].value < padding:
            break
        current_input_seq.append(inputs[i])
        if offset == 0:
            sequences.append({
                inputs: current_input_seq,
                outputs: current_output_seq
            })
            current_input_seq = []
            current_output_seq = []

    for output in outputs:
        output.color = "uncolored"
        output.colorvalue = 0

    for seq in sequence:
        col = seq.inputs[0].color
        for input in seq.inputs:
            if input.color != col: col = "undefined"
        for output in seq.outputs:
            output.color = col
            output.colorvalue = output.value - padding
```

Evaluation:

- **Simplicity**: the algorithm is considerably more complex, and less intuitive, than plain OBC. Furthermore, creating transactions with more outputs than inputs (or vice versa) adds an extra layer of complexity, as a transaction may need to have an extra input or output to pay for or collect excess padding.
- **Non-restrictiveness**: in some ways, padded OBC is actually slightly less restrictive than plain Bitcoin, as it becomes practical to send a single satoshi of a given color. On the other hand, the unusability ofSIGHASH_ANYONECANPAY still applies.
- **Atomic tradeability**: atomic trades are very easy,
- **Efficiency**: padded OBC wins over traditional OBC on one count: it can sometimes detect non-colored-coin transactions, thereby in many cases prematurely ending the search. This usually happens when, in non-colored-coin transactions with multiple inputs and outputs, the number of inputs and outputs does not match up, leading to the coloring algorithm discovering no sequences. For even greater efficiency, one can always add

more restrictions, for example outright mandating that the number of inputs and outputs must be the same, and forbidding outputs smaller than the padding. Aside from this, however, the fact that a potentially exponential upward search may be required still remains.

- **Space efficiency**: OBC transactions take up exactly as much space as regular Bitcoin transactions.

Tagging-based coloring

Tagging-based coloring is a different approach to coloring: rather than trying to determine the color of inputs and outputs by their order, simply explicitly state what the color is in each output. There are several places in a Bitcoin transaction where one can potentially encode metadata:

- Low-value digits in the output value (eg. 0.00100057 vs 0.00100043)
- Sequence number on inputs
- Additional data-carrying outputs using OP_RETURN

Algorithm for sequence number-based coloring:

```
for output in outputs:
       output.color = NULL
       output.colorvalue = 0

   for input in inputs:
       destinations = []
       for j in [0 ... length(outputs) - 1]:
           if [input.sequence / (2 ^ j)] % 2 == 1:
   destinations.append(outputs[j])
       total_outputvalue = 0
       for output in destinations:
           total_outputvalue += output.value
       for output in destinations:
           if output.color == NULL: output.color = input.color
           if output.color != input.color:
               output.color = "uncolored"
               output.colorvalue = 0
           else:
               output.colorvalue += floor(input.colorvalue *
   output.value / total_outputvalue)
```

Essentially, the algorithm works by using the sequence number as a bit field, so if bit 7 of input 4 is set to 1 that means that input for maps to output 7. An input can map to multiple outputs; in that case, the input's colorvalue is shared proportionally between the outputs. For example, if input 3 (with 400 blue satoshis) maps to outputs 2, 5 and 7, and output 2 has 200 blue satoshis, output 5 has 500 and output 7 has 700, output 2 gets

floor(400 * 200 / 1400) = 57 blue satoshis, output 5 gets floor(400 * 500 / 1400) = 142 and output 7 gets floor(400 * 700 / 1400) = 200. Note that in many cases single satoshis will be deleted due to rounding errors; it's the job of the client to avoid this if it is a problem.

The algorithm for value-based coloring is the same, except in reverse: the binary digits of the output value are used to determine which inputs map to a given output, and then the sharing algorithm is identical.

Evaluation

- **Simplicity**: the algorithms are somewhat harder to implement than order-based algorithms, especially because special care would be required to prevent accidental satoshi destruction due to rounding errors.
- **Non-restrictiveness**: all of these protocols have one weakness: they seriously limit the number of inputs and outputs that a transaction can have. With sequence number OBC, transactions can have at most 32 outputs, with value-based OBC ~10-13 outputs, and with output-data based OBC 12 outputs. It is possible to expand this to a few hundred or thousand by limiting the number of outputs per input and using combinatorics-based encoding algorithms, but this would impose another restriction and is overcomplicated. Furthermore, sequence number OBC has the limitation that it limits the effectiveness of advanced transaction replacement-based protocols, as the low-order digits of the sequence number would be constrained, and value-based OBC has the limitation that it imposes constraints on the low-order digits of transaction output values.
- **Atomic tradeability**: atomic trades are easy.
- **Efficiency**: the algorithm will nearly always be able to detect and avoid processing uncolored transactions, so the upward search will be limited to transactions of a given color. This essentially provides the maximum possible efficiency with any output-based coloring scheme.
- **Space efficiency**: the sequence and value-based algorithms cost no extra data.

Alternative Tagging-based Coloring

There is also an alternative coloring algorithm, which uses tagging to determine not the permutation, but rather the coloring itself. The last two digits of the output value are used to represent the color of the output. However, because there is a risk of collisions (in fact, with 101 colors at least two must have the same two-digit ID), we assign to each color ten two-digit IDs, and use the third digit in the output value to let the transaction builder choose an index.

The version described here is a formalization and minor modification of an algorithm described by Paul Snow.[8]

```
colors = []
   for input in inputs:
       colors.push(input.color)
   for output in outputs:
       high = floor(output.value / 100) % 10
```

```
        low = output.value % 100
        output.color = "uncolored"
        if low > 0:
            for color in colors:
                for i in [0 ... 9]
                if SHA256(get_color_definition(color)+i) == low:
                    output.color = color
                    output.colorvalue = output.value
```

Unlike the other algorithms, this one does need to be followed up by a value consistency check:

```
success = 1
   cv = {}
   for color in colors: cv[color] = 0

   for input in inputs: cv[input.color] += input.colorvalue
   for output in outputs: cv[output.color] -= output.colorvalue

   for color in colors:
       if cv[color] < 0:
           success = 0

   if success == 0:
       for output in outputs:
           output.color = "uncolored"
           output.colorvalue = 0
```

- **Simplicity**: the algorithm is arguably even simpler than plain order-based coloring.
- **Non-restrictiveness**: one could maliciously construct a color which cannot be exchanged for a given other color in 10^20 steps (or two colors which are mutually incompatible in 10^10 steps). Also, not all transaction sizes are possible. However, statistical tests show that any amount of more than 2000 satoshis can nearly always be built from multiple outputs[9].
- **Atomic tradeability**: atomic trades are easy.
- **Efficiency**: the algorithm will nearly always be able to detect and avoid processing uncolored transactions, so the upward search will be limited to transactions of a given color. This essentially provides the maximum possible efficiency with any output-based coloring scheme.
- **Space efficiency**: no extra data cost required.

Per-Satoshi Tracking

Per-satoshi tracking is another form of order-based coloring, but which is very limited in

scope. Unlike OBC or padded OBC, which seek to assign colors to outputs, per-satoshi tracking assigns a color to each satoshi in an output.

Algorithm:

```
def getParent(tx, index, offset):
       total_offset = 0
       for i in [0 … index-1]: total_offset += tx.outputs[i].value
       total_offset += offset
       j = 0
       while total_offset > tx.inputs[j].value:
           total_offset -= tx.inputs[j].value:
           j += 1
       return (get_previous_tx(tx.inputs[j]), j, total_offset)
```

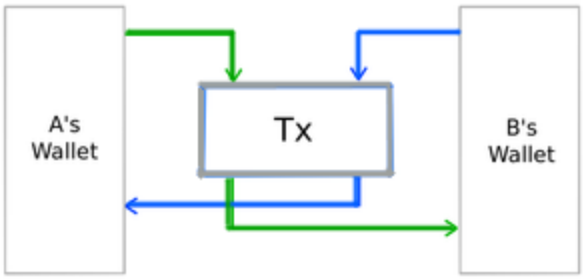
getChild works similarly but in reverse.

The main advantage of this is efficiency: there is no risk of exponential blowup, as each satoshi has only one parent. However, for most cases this protocol also introduces an impractical amount of inefficiency, as most outputs have many thousands, or even millions, of satoshis. Thus, this is a special-purpose protocol, limited to unique colored coins such as collectibles and smart property. However, for that purpose, it provides an incredibly useful and light-weight implementation of colored coins.

Since individual satoshis cannot practically be sent due to the dust limit, a colored satoshi would likely come pre-packaged with 10000 satoshis of padding. A genesis transaction would, by convention, publish one satoshi per output, always at offset zero, and then the satoshi would be passed around along with its padding.

Decentralized Exchange with Colored Coins

One beneficial feature of colored coins is the ability to easily do atomic swaps - that is, trade X units of coin A for Y units of coin B without either party needing to trust the other. The idea behind this is simple: have a multi-color transaction with an input of X units of coin A from party 1 and Y units of coin B from party 2, and an output of X units of coin A to party 2 and Y units of coin B to party 1. With OBC or padded OBC, this simply consists of putting the inputs and outputs in order - input and output 0 for coin A and input and output 1 for coin B.



Here is a more precise example in plain OBC, including a 10000-satoshi fee:

```
{
        inputs: [
            { color: 'green', address: [A], value: 60000 },
            { color: 'blue', address: [B], value: 20000 },
            { color: 'uncolored', address: [B], value: 70000 }
        ],
        outputs: [
            { color: 'green', address: [B], value: 60000 },
            { color: 'blue', address: [A], value: 20000 }
            { color: 'uncolored', address: [B], value: 60000 }
        ]
    }
```

However, this is not enough to make a full exchange; a full exchange requires the ability for users to post orders (eg. "I want to sell up to 50 X at a ratio of at least 3.3 Y for 1 X") and have those orders be enforceable. If an order is not enforceable, then anyone can spam the order book with false orders and thereby either manipulate or shut down the market outright. With colored coins, unfortunately, orders are not enforceable - anyone can make an order and refuse to sign any specific exchange transactions. Partially signed transactions with SIGHASH_ANYONECANPAY are powerless to solve the problem in most cases; if you are trying to sell colored coins for BTC, it is a viable solution, but if you are trying to buy colored coins anyone can defraud you by sending in uncolored coins, which the blockchain cannot distinguish from colored coins. Thus, a centralized, or at least semi-centralized, service is still required to act as a spam filter to make colored coin exchange a reality. However, the level of centralization is much less than between Bitcoin and other currencies, as the centralized service does not need to actually store any funds - the only task that it needs to perform is that of screening users to see if anyone is making false orders, and then forbid them from making further orders.

References

1. http://nwotruth.com/wikileaks-boycotts-too-big-to-fail-with-bitcoin/ Wikileaks boycotts: too big to fail with Bitcoin
2. https://github.com/bitcoin/bitcoin/pull/2738 Bitcoin 0.9 change
3. https://github.com/bitcoinx/colored-coin-tools/wiki/colored_coins_intro Colored coins intro (describes OBC and POBC)
4. https://coloredcoins.org
5. https://github.com/killerstorm/BitcoinArmory/blob/color/p2ptrade.py P2Ptrade
6. https://bitcointalk.org/index.php?topic=112007.0 Discussion on atomic coin swapping
7. https://en.bitcoin.it/wiki/Smart_Property Smart property
8. http://brownzings.blogspot.com/2013/06/an-alternate-more-simple-implementation.html An Alternate More Simple Implementation of Colored Coins, Paul Snow
9. http://vbuterin.com/coloredcoins/valuecalc.py Dynamic programming algorithm for

sending a desired number of satoshis under value-based coloring

1.