Bitcoin Colors — A New Proposal DRAFT

Gideon Greenspan — February 2014

A full implementation of this paper is now available at http://coinspark.org/
Please note that some of the specific technical details (e.g. bit encoding) have changed.

Background

Bitcoin has already been proven as a safe, fast and cheap means for transferring ownership across the Internet without a central authority. While much bitcoin activity is driven by speculation or investment in bitcoin for its own sake, it is also being used for many transactional purposes, most recently being accepted for orders at Overstock.com in January 2014.

When thinking about bitcoin, it is important to separate out several layers:

- Bitcoin as an asset. Relative to currencies such as the US dollar, bitcoin increased in value approximately 100x during 2013, leading to much interest amongst the media, speculators and investors. The rise in bitcoin's value is caused by a combination of recognition of its technological potential and its nature as a decentralized asset ("Gold 2.0") which cannot be inflated or seized by governments.

- Bitcoin as a network. As of January 2014, the bitcoin network consists of approximately 10,000 nodes (source: getaddr.bitnodes.io) which communicate with each other in a peer-to-peer fashion, exchanging information about transactions and confirmed blocks.

- Bitcoin as a piece of software. The majority (if not all) of the nodes on the bitcoin network are running the original bitcoin client, early versions of which were written by the mythological Satoshi Nakamoto, and which continues to be developed as open source.

- Bitcoin as a protocol. The protocol, as designed by Satoshi Nakamoto and others, is described extensively on the Bitcoin wiki at en.bitcoin.it. It defines exactly which messages must be communicated between nodes in the peer-to-peer bitcoin network, what constitutes a valid transaction, how the network mining difficulty is calculated, etc...

- Bitcoin as a technology. Most journalists writing about bitcoin miss the fundamental point that it represents a significant technological advance in the field of computer science. The problem (known as Byzantine Generals) it addresses is this: A peer-to-peer network contains a large number of nodes exchanging information with each other, with no central authority. A significant minority of these nodes can be faulty or (more significantly) colluding and malicious. How do we ensure that all the honest nodes in this network share an identical view of the world, with an extremely high probability, given sufficient time?

Each layer builds upon the one below. It is quite possible that some of these layers of bitcoin will succeed, while others fade into irrelevance. For example:

- A more efficient version of the bitcoin software is created and gradually replaces the original bitcoin client across the bitcoin network.

- One of the 'altcoins' such as Litecoin replaces bitcoin as the dominant cryptocurrency network, so that only the technology and significant parts of the protocol remain important.

- The value of a bitcoin crashes back down to 2012 levels, but the bitcoin network becomes an extremely popular way to transmit and confirm transactional information across the Internet.

The goal of this document is to describe how bitcoin the technology, protocol and network can be extended to allow the exchange of various types of digital asset. Crucially, this extension will require no changes to the bitcoin software, beyond those which are already planned for version 0.9 (source: https://bitcoinfoundation.org/blog/?p=290). In the short run this extension might well lead to a lowering of the value of a bitcoin, since it allows small quantities of bitcoin to represent items of much larger value, reducing the number of bitcoin required per transaction, and thus reducing demand for bitcoin itself. However if bitcoin colors succeed and the transaction rate of the bitcoin network is increased as necessary (current limit: ~300,000 per day), in the long run the network could permit a very large number of digital transactions to take place quickly and cheaply. This would in turn drastically increase demand for bitcoin as an asset.

In our terminology, each digital asset issued over the bitcoin network is called a 'color'. In a marked departure from bitcoin the asset, each color has a single backer or issuer, who is contractually obliged to give something specific in exchange for units of that color. Colors could represent a license to access some content, credits for an online service, air miles, coupons, etc… (While I recognize the ideological desire of many early adopters of bitcoin to replace fiat currencies, one shudders to think of the social consequences for modern society. In any event, this seems extremely unlikely to happen and echoes predictions made during the 1990s that e-commerce would put an end to brick-and-mortar stores.)

Before continuing I would like to acknowledge the work of many others in the bitcoin community in developing and implementing the notion of coin coloring. This work[1][2][3][4][5] can be found from the coloredcoins.org website as well as various forums such as bitcointalk.org and the Colored Coins Google group (https://groups.google.com/d/forum/bitcoinx). While the idea of colored coins has been "in the air" for at least a year, I have not yet seen a proposal which I feel addresses the core requirements for a successful consumer-oriented coloring protocol. Nonetheless this may well be due to the insufficiencies of my own research, in which case I apologize in advance for omitting the appropriate attributions to previous work, and give full credit where it is deserved.

Finally, it should be noted that the term 'colored coins' is a misnomer for the proposal described

[1] https://docs.google.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE
[2] https://github.com/mastercoin-MSC/spec
[3] http://brownzings.blogspot.com/2013/06/an-alternate-more-simple-implementation.html
[4] https://github.com/bitcoinx/colored-coin-tools/wiki/colored_coins_intro
[5] https://bitcoil.co.il/BitcoinX.pdf

herein, even though it shares some similarities with other bitcoin coloring proposals. To underline the difference I am calling the concept ‘Bitcoin colors’. Under this proposal, in no way are units of bitcoin being assigned colors that move together with those coins across transactions. Rather, each input and output of a bitcoin transaction can have a certain quantity of one or more colors attached to it, alongside the regular quantity of bitcoin. Each color moves independently from inputs to outputs by applying the appropriate rules, and no color’s movements are affected by what happens to any of the other colors. This allows each color’s movements to be tracked completely separately, whether forwards or backwards across the chain of transactions. Recall that the ‘balance’ of a bitcoin address is effectively the total of the unspent bitcoin transaction outputs which the private key of that address can unlock. The balance of a bitcoin address in each color is the total amount of that color in those same unspent outputs. Small quantities of bitcoin act as a kind of “stamp duty” to enable transfers of color within transactions, but they are not colored themselves.

Principles

I believe that a successful consumer-oriented protocol for bitcoin colors must fulfill these criteria:

- Full compatibility with the bitcoin protocol and network. This will enable transfers of color value to leverage the size and resilience of the network as it currently stands.

- Safe to interact with existing bitcoin wallet software. Since regular bitcoin wallet programs will not recognize bitcoin colors, and cannot transact in them safely, it should not be possible to send colored assets from color-aware wallets to non-color-aware ones. This point is especially crucial because early adopters of color-aware wallets are likely to try sending some color to other bitcoin users they know.

- Preservation of color quantities. Apart from an initial transaction which creates the units of color, it should not be possible for any transaction to create or destroy units of a particular color. All color units in a transaction’s inputs flow to one of its outputs. Note however that is always possible to “destroy” color units (and indeed bitcoins) by sending them to a fake bitcoin address, for which the private key is not known.

- Low computational burden on color-aware bitcoin wallet software. The responsibility for tracking transactions of each color cannot lie in the bitcoin network, because the network is not (initially, at least) aware of colors in transactions. It cannot lie in bitcoin wallet software, because the software might need to backtrack through an exponentially increasing number of transactions in order to check the history of a particular color. As will be detailed later on, I believe that the day-to-day responsibility for tracking transactions of a particular color should lie with the issuer/backer of that color, or an external service that they employ. But this does not take us back to centralized asset tracking, since the bitcoin blockchain serves as a comprehensive and public audit trail that enables disputes to be objectively resolved. For users the key difference is that the backer cannot enforce a fee on transactions involving the asset, cannot freeze or seize the asset, and cannot decide who has the right to transact with whom, since in most cases it won’t be possible to tell.

- Compatibility with lightweight bitcoin clients. As of February 2014, the bitcoin blockchain is over 14 GB in size, and growing at a rate of almost 1 GB per month (source: https://blockchain.info/charts/blocks-size). While various proposals discuss how to reduce this size, none are yet being incorporated into the bitcoin software. In the meantime as the network's transaction rate increases, the blockchain's growth will further accelerate. As a result most end users of bitcoin will not wish to store a copy of the entire blockchain on their own computer or mobile device, but will prefer to run lightweight clients such as Electrum and Multibit. These store only a small amount of crucial information on the user's computer and rely on the bitcoin network or special-purpose servers to fill in the gaps.

- Smooth transition path to a color-aware bitcoin network. As mentioned previously, the bitcoin network is currently unaware of colors, and this leads to the requirement for each color to have its own tracking server, to avoid placing a high computational burden on users of that color. Nonetheless the protocol should be designed to enable the bitcoin network to become color-aware in future, via a smooth transition in which nodes are gradually replaced. This transition should require no hard fork in the bitcoin protocol and no architectural changes to the bitcoin network or transaction model. An immediate consequence is that color balances should preferably be attached to transaction outputs rather than bitcoin addresses, since this is the way in which units of bitcoin (the asset) are moved around.

- Low byte-count burden on the bitcoin blockchain. Each bitcoin transaction can have an optional transaction fee attached, which is collected by the miner who successfully incorporates that transaction into a block. While it is currently possible to have zero-fee transactions confirmed, this situation is not expected to last long. As with any common and limited economic good, the bitcoin blockchain will end up being abused if there is no cost attached to storing information on it. Initiatives such as MasterCoin have already proven that it is possible to add additional non bitcoin related information to transactions, and version 0.9 of bitcoin will explicitly enable up to 40 (not 80, as originally announced) additional bytes to be associated with each transaction. Since this can be used by any party to create a globally replicated store of a crucial piece of information, such as the hash of a contract, demand at zero cost is essentially infinite. Therefore the cost is expected to rise, until demand and supply (of bandwidth and storage space on the nodes in the network) are matched. Consequently there will be a cost associated with each additional byte which is added to a bitcoin transaction and our protocol should aim to use as few bytes as possible. (As an aside, this makes it clear that bitcoins do indeed have intrinsic value – they grant the holder a certain proportion of the global bidding power for embedding data in the blockchain, a decentralized globally distributed repository of information. As a further aside, it seems inevitable that the blockchain will eventually be filled by whichever transactions provide the highest economic value per byte, since the issuers of those transactions will be willing to pay the highest fees. These are unlikely to be payments for cups of coffee unless the network capacity is hugely expanded.)

- No limit on the number of color issues. Since the entire point of bitcoin colors is to enable different types of digital assets to be transacted on the bitcoin network, we would not wish to introduce a limit on how many different colors can ever be created. Note that this stipulation does not introduce a scalability problem, because parties making use of bitcoin colors will only need to track those colors which are of interest to them. As mentioned previously, under this proposal the paths taken by one color are completely independent of the paths taken by other colors. Perhaps services will arise that track every single color ever issued on the network, but this will not be necessary for coloring to succeed. On a related point, we want to avoid the need for namespace synchronization, where different parties issuing bitcoin colors must coordinate in order to ensure that their color identifiers do not clash.

- Non-dependency between the movements of each color. If the notion of colors succeeds, there will likely be a large number of different colors being transacted simultaneously over the bitcoin network, for different purposes, among different communities of users. As mentioned previously, the day-to-day responsibility for tracking balances of each color should lie with a color server, which is maintained by the color issuer. We want to enable these servers to track the movements of those colors which are of interest to them, without needing to follow the movements of any other color flowing over the network.

- Automatic color discovery. Once two parties are using color-aware bitcoin clients, one party should be able to send some color to the other without requiring the recipient to define in advance which colors they are able to receive. This is a key point in terms of usability, so that all the sender needs is the recipient's color-aware bitcoin address, rather than also needing to verify that the recipient is able to receive assets of a certain color. Color transfers should come with sufficient accompanying information in order to allow the recipient to understand what they have been sent and decide whether it is of value. (One might be concerned about "color spam" under such a scheme, but so long as each transaction has some non-negligible cost, color spammers will quickly go out of business.)

- Connecting colors to contracts. As the holder of a bitcoin color asset, I should have watertight legal protection and proof of the promises that have been made by the backers of that asset. I should be able to store everything required to present that proof, even if the issuer/backer attempts to renege on those promises and disappears from the scene. This must be sufficiently strong to hold up in a regular court of commercial law. Note of course that nothing can protect the holder from bankruptcy of the color issuer, but in this sense an asset represented by a bitcoin color is no worse off than the same asset in traditional form.

- Communication between an asset issuer and those holding the asset. There should be some simple method for the dissemination of notifications, warnings, news, etc…, by the issuer of the color to all parties which are holding units of that color. (As an aside, one byproduct of bitcoin coloring schemes is that an issuer will be able to automatically send a new asset to all those holding an existing asset, over the bitcoin network. They could also collect feedback from asset holders by having that feedback signed by a bitcoin private key.)

To repeat what was said in the introduction, I have not yet seen a protocol definition which fulfills all of these criteria, and this is what motivated me to create a proposal of my own.

Color-aware addresses

A primary concern for any bitcoin coloring scheme should be how it interacts with non-color-aware bitcoin wallets. The problem is this: Alice has a color-aware bitcoin wallet, and sends some e-commerce credits to Bob along with a minimal amount of bitcoin. Bob receives the bitcoin, but does not see the credits, since his wallet is not color-aware. Now Bob sends some bitcoin to Charlie, and the corresponding transaction happens to spend the output of the payment from Alice to Bob. So Bob has just sent Alice's credits to Charlie, without being aware of it. Alice calls Bob later to ask if he received the credits. Bob expresses surprise that this can be done over bitcoin, so Alice says "sure it can - you just need to upgrade your wallet". Bob does so in excitement, but the credits are gone, and now he has to go chasing after Charlie in the hope of getting them back. The credits may well have been passed on many more times by then.

A previous version of this proposal suggested that this problem can be avoided by assigning color quantities to bitcoin addresses, rather than transactions. Any color not explicitly transferred out of an address would still belong to that address, even if the transaction output that brought in the color has already been spent. While this solution is workable and solves the core problem, it has several shortcomings (thanks to Jeremy Spilman for pointing them out):

- It is a departure from the transaction-oriented model of bitcoin and therefore will be harder to implement using existing bitcoin tools and source code.

- It prevents safe peer-to-peer exchanges between two colors, or between one color and bitcoin. This is because two individuals who wish to perform an exchange between specific quantities of two colors cannot tie the inputs of that exchange to the specific outputs of previous transactions, whose color quantities are known.

- It greatly increases the complexity of backtracking to find the history of a color through the transaction chain, since all transactions of that color on the blockchain must be examined instead of only those which are on the converging tree back from the current transaction to the transaction in which the color was created.

The proposed alternative is for color-aware wallets to use a different representation of their receiving addresses compared to regular bitcoin wallets. This allows a bitcoin address to implicitly declare whether it is ready to receive and transact safely in colors. A color-aware wallet will not attempt to send color to an address which is not in this color-aware format. Since regular bitcoin wallets will not recognize the color-aware address format, color wallets must also provide their addresses in regular bitcoin format, which can be used for sending them bitcoin only. While this solution has some shortcomings in terms of usability, these would hopefully drop away over the long term as most wallets become color-aware, and there will no longer be a need for the old bitcoin-only address.

Color-aware address representations should be instantly recognizable, for example, beginning with the character C. They should map 1:1 with regular bitcoin representations, but this mapping should not be trivial for a person to apply, since that would enable users to easily work around the restriction. One simple suggestion is to reverse the order of the bytes in the binary bitcoin address before converting it to Base58Check encoding. Such an address, even if had the C prefix removed, would not be accepted by traditional bitcoin clients, because of the 4-byte checksum within. (Actually 1 in 2^{32} of these reversed address will look like valid bitcoin addresses but that is a level of risk we are probably willing to accept.)

Transaction model

The proposed model of color transactions covers two events – the act of color genesis (creating a new color in the network) and the transfer of color from one transaction to another.

Color genesis

Color genesis occurs by a transaction in which the issuer sends an minimal quantity of bitcoin from one address to another. (Unlike many other colored coins proposals, the quantity of bitcoin is not relevant – it should merely be sufficient to ensure the transaction is permitted onto the blockchain under bitcoin's anti-dust policy.) The end result is that the first output of the genesis transaction contains all of the created color, which can then be sent elsewhere.

In order to be considered valid, a color genesis transaction must include a bitcoin transaction fee (to be collected by the miner) which is at least as large as the smallest bitcoin value of any of its outputs. The purpose of this is to prevent bloating of the blockchain with excessive colors, as will be explained in more detail later. Any color genesis transaction not fulfilling this criterion will fail to issue its color.

Five additional pieces of information are attached to a genesis transaction, in the extra data field permitted from bitcoin 0.9:

1. A special short string which identifies this transaction as a color genesis.
2. The URL of an HTML color specification page which provides information about the color in human-readable form, and which contains additional information about the color in special metadata fields. This information shall include at least: a name, longer description, a link to the contract for the color, the URL of the server/s responsible for tracking the color, and the (negative or positive) power of 10 by which a quantity of units should be multiplied before presenting it to the user.
3. The (integer) quantity of color units which are being created.
4. A cryptographically secure one-way hash of the color name, description, display multiple and contract, to enable holders of that color to prove the promises which underlay it at the time of issuing. (The display multiple is included in the hash, so there is no chance of an issuer changing how many units a user can see they are holding.)

Note that only self-contained formats should be permitted for the color contract. Most pointedly, HTML should not be allowed, since the contents of a web page as viewed by a user can be changed by modifying externally linked assets such as images or Javascripts which were not included in the hash calculations. An initial set of suitable formats could be: PDF, UTF-8 encoded plain text, JPEG and PNG. Since PDF files can also embed external resources, their content will need to be checked to prevent this.

Color transfer

A transfer of color between parties occurs via a regular bitcoin transaction which contains inputs and outputs. As with color genesis, the quantity of bitcoin moved by the transaction itself is not relevant, so long as it is sufficient for the transaction to be relayed and confirmed. As with units of bitcoin, for each color, the inputs of a color transfer transaction are 'spent', so that any color which is meant to remain with the sender must be returned in the form of change. Recall that for bitcoin, any quantity which is not assigned to an output is considered as the transaction fee to be collected by miners. For color transfers we apply a different policy. Any quantity of a color which is not assigned explicitly to an output is transferred to the last output automatically. If a transaction does not intend to send color, the last output can simply be used as the 'change' destination, and all color units in the transaction inputs will remain with the current owner. (A previous version of this document specified that the first output should be the default destination for colors, but this was changed because the last output seems to be more commonly used for change in regular bitcoin clients.)

In order to be considered valid, a color transfer transaction must include a bitcoin transaction fee (to be collected by the miner) which is at least as large as the smallest bitcoin value of any of the outputs, multiplied by the number of color transfers sent to outputs other than the last. The purpose of this is to prevent bloating of the blockchain with excessive unspent transaction output colors, as will be explained in more detail later. Any color transfer transaction not fulfilling this criterion will effectively fail and be treated like an unmarked bitcoin transaction, sending all of the color in its inputs to the last output only.

Any transfers of color to an output other than the last are specified explicitly as follows:

1. A special short string which identifies what follows as a list of color transfers.
2. The list of individual color transfers, each of which contains:
    a. A reference to the corresponding color genesis transaction. To save space this reference will take the form of the block number of the genesis transaction and the byte offset of the genesis transaction within that block. (This introduces a natural delay between color issue and transfer, which has the additional side benefit of creating a 'cooling off period' for accidental or unauthorized issues.)
    b. A prefix of the ID of the color genesis transaction referenced above. This avoids the (unlikely) situation where a color transfer in a transaction is misinterpreted due to a change in the history of the blockchain prior to that transaction, which causes the block number and byte offset to reference a different color genesis from the one intended by the sender.
    c. The index of the input or inputs within the bitcoin transaction from which that color is being transferred. (In theory we don't need to specify this, since all quantities of a color are fungible, but moving color from specific inputs only enables backtracking for a color to be performed much more efficiently.)
    d. The index of the output or outputs to which that color is being transferred.
    e. The number of color units being transferred to each output. If the specified inputs of this transaction do not have sufficient balance of the color for all of the outputs, the outputs receive the specified quantity of color in order until all of the available units have been transferred. (The last output to receive some color may well receive less than the quantity specified.) Color transfers will also be performed under this principle in the order in which they appear within the list of color transfers, i.e. first come, first served. It should be noted that this means the output quantities of a color transfer

cannot be trusted without checking the converging tree of transactions all the way from this transaction back to the appropriate color genesis transaction. (If we find one path when backtracking through the tree which conveys a sufficient quantity of the color, we can stop checking, since other paths can only increase that amount.)

After performing the specified transfers of color, any remaining quantities of any color from the inputs are transferred to the last output as change.

Note that because bytes in the blockchain are at a premium, the list of color transfers should be represented as efficiently as possible, for example using an additional metadata byte to specify how many bytes will be used for the representation of each field. In addition it should be possible to skip certain fields in order to save more space. For example:

- No reference to color genesis transaction. If in the first item, this is not valid, otherwise use the same color as the previous item in the list.
- No reference to the quantity of color. If in the first item, assume a value of 1, otherwise use the same quantity as the previous item.
- No reference to the index of the inputs. If in the first item, assume the first input only, otherwise use the same inputs as the previous item.
- No reference to the index of the outputs. If in the first item, assume the first output only, otherwise use the same outputs as the previous item.

By using this type of encoding, many color transfers after the first in the list can be represented by just two bytes, enabling many to fit within byte limit. This will be crucial for transactions which are sending units of color to many outputs at once, or transferring many colors between a specific input and output. (By extension, special codes in the metadata byte could indicate “next input”, “next output” or “all inputs”, meaning that a long list of transfers could be represented by a single byte each. Once it is observed what types of color transfer lists are most popular, it will no doubt be possible to come up with many more possibilities for efficient encoding.)

Minimum transaction fees

As described above, both color genesis and color transfer transactions have minimum bitcoin transaction fees, which are based on the smallest bitcoin value of their outputs. The purpose of this constraint is to prevent the blockchain being polluted with excessive color issues, or an excessive number of unspent transaction output colors (thanks to Flavien Charlon for pointing out this issue). Excessive use of colors would not be a problem initially, since the network is color-blind and each color is tracked by its own server. However we hope that the bitcoin network will become color-aware in future, so that nodes will track the color balances of transaction outputs alongside their bitcoin balances. We want to ensure that users cannot behave now in a way that will present scalability problems to a future color-aware network.

First let’s examine how the bitcoin network dealt with a similar problem in the past. Since version 0.8.2 of bitcoin, any transaction which contains an output lower than 5430 satoshis (i.e. 0.0000543 BTC) will not be forwarded by nodes running the default settings. This “anti-dust” policy prevents users creating excessive unspent transaction outputs (UTXOs), which must be kept in memory by bitcoin nodes in order to allow new transactions to be verified quickly. In other words, users must pay a minimum fixed ‘deposit’ for each unspent transaction output that nodes keep in memory. Users who collect a transaction output themselves can recover this deposit, but only by creating transactions which to reduce the total number of UTXOs to be tracked.

Our scheme for minimum color transaction fees piggybacks on bitcoin’s anti-dust restrictions in order to apply a similar philosophy. We assume that there will be a linear cost in terms of memory for each different color on each unspent transaction output that nodes in a color-aware network will keep in memory. A color genesis transaction creates one unspent transaction output color (UTXOC), so its issuer must pay a transaction fee to miners which is at least as large as the bitcoin value of its smallest output. (This in turn must be larger than the regular bitcoin anti-dust threshold prevailing at the time otherwise the transaction will not be verified and relayed by the network.) A color transfer transaction creates up to one UTXOC per color that it transfers to an output other than the last. Therefore its issuer must pay the smallest bitcoin value of its outputs multiplied by the number of such transfers. Note that if a color transfer transaction is sending all of the color in its inputs to the last output, it cannot create additional UTXOCs, so no minimum transaction fee is required. (Unlike the minimum output size in regular bitcoin, these fees cannot be collected back by the user, since a UTXOC is not necessarily removed when combined with other colors in one output. In theory a miner could abuse the system by issuing color transactions and attempting to collecting these fees, but on average they would receive back less than 50%, unless the entire bitcoin system had been compromised by a 51% attack.)

By tying the minimum fee for color transactions to the bitcoin value of the smallest output, we are making users pay the current bitcoin anti-dust threshold of 5430 satoshis for each UTXOC created. It can be assumed that this threshold will change (probably downwards) in future, so the minimum fee for color transactions will automatically change in lock-step. Nonetheless there will always be a canonical answer as to whether a particular color transaction is valid, independent of any bitcoin’s node anti-dust threshold at the time, because the criterion is based only on the relationship between the transaction fee and the smallest bitcoin output.

Transaction formats

This section gives a more detailed technical description of how the additional 40 bytes of information permitted from bitcoin version 0.9 could be used to represent the details of color transactions. (The size limit was originally announced as 80 bytes, but was reduced in the latest modification of the bitcoin source code on Github, as of February 26, 2014.) The precise details in this section are not crucial – the main goal is to demonstrate that it is possible to represent all of the required information within the limited space available.

Color genesis

Field description	**Size**	**Content (example or specification)**
Color genesis identifier	4 bytes	`COCO` i.e. `0x43 0x4f 0x43 0x4f`

Color quantity of units created (mantissa and exponent)	2 bytes	17 × mantissa between 1 and 923 + exponent (base 10) between 0 and 16. This implies a maximum value of 15691 (out of 65535). Any other values are considered invalid for now but could act as markers for other color genesis transaction formats in future.
Domain where color specification page is stored	≤22 bytes	Domain name encoded with 2 bytes per 3 characters – see encoding scheme below.
Hash of key information from color specification page	remaining bytes up to limit of 40	prefix of sha256(name + “\n” + description + “\n” + multiple + “\n” + content of contract) up to size limit, keep leading zeros

The combination of the color quantity mantissa and exponent allows any quantity of color between 1 and 9.23×10^{18} to be represented with three significant digits of accuracy within just 2 bytes. Note that this is sufficient to reach the highest 64-bit signed integer 9.22337×10^{18}. The quantity should be interpreted subject to this upper bound.

Domain names are encoded with 2 bytes per 3 characters according to the following scheme. Each pair of adjacent bytes is interpreted as an unsigned 16-bit integer, stored in small-endian order. That integer, which we shall call *x*, must have a value between 0 and 63999, otherwise it is considered invalid. Three ordered numbers between 0 and 39 are extracted from that integer according to the following formulae:

1. *x* modulo 40
2. (round_down(*x* ÷ 40)) modulo 40
3. round_down(*x* ÷ 1600)

Each of these numbers between 0 and 39 is interpreted as follows:

- 0 … 9 = the digits `0` to `9` in numerical order
- 10 … 35 = the letters `a` to `z` in alphabetical order
- 36 = `-` (hyphen)
- 37 = `.` (period)
- 38 = invalid
- 39 = end marker

Domain names are decoded by repeatedly fetching pairs of bytes and interpreting them according to this scheme. As soon as the end marker is observed in the generated sequence of characters, decoding stops. No matter where the end marker appears in the set of 3 characters decoded from a 2-byte integer, both bytes are considered “burnt” in the decoding stream.

The URL of the color specification page is defined as the concatenation of:

1. `http://`

2. The domain in the color genesis transaction.
3. `/bitcoin-color-`
4. The first 16 characters of the lower case hexadecimal transaction ID of the color genesis.
5. `/`

For example the domain `mysite.com` could be converted to:

```
http://mysite.com/bitcoin-color-f5d8ee39a430901c/
```

This purpose of this scheme is to enable the URL of the color specification to be defined mostly by implication, without taking up too many bytes in the color genesis transaction. By insisting that the color specification page is at the root of the domain, we avoid situations where users of basic page hosting services can make their color issues appear to be backed by that hosting service. (Placing a specific file in the root directory of a site is one of the methods that Google Webmaster Tools uses to establish site ownership, so it is reasonable to assume that it is safe.) See the later section for information about the content of the color specification page.

The longest possible prefix of the hash of the key information (name, description, contract) is included at the end of the genesis transaction metadata, subject to the overall limit of 40 bytes. This effectively means that, the shorter the domain name of the color specification page, the more space remains for this hash prefix. Longer hashes have the advantage for users that they make it harder for color issuers to retroactively change the name, description and/or contract of their color. A short domain name like `ebay.com` can be encoded in just 6 bytes, leaving 28 bytes for this hash, which is close to the theoretical limit of 32 bytes that the sha256() hash function can offer.

Color transfer

Field description	**Size**	**Content (example or specification)**
Color transfer list identifier	4 bytes	`coco` i.e. `0x63 0x6f 0x63 0x6f`
Bytes in list of color transfers (not including identifier and this byte)	1 byte	`12` (unsigned 8-bit integer)
List of color transfers	≤35 bytes	see below

By including the byte count in the header, the list of color transfers can be skipped by software which is not interested in its contents. The remaining bytes not used by the list of color transfers can be used for additional information relating to the transaction.

Each item in the list of color transfers takes the following format:

Field description	**Size**	**Content (example or specification)**
Packing metadata	1 or 2 bytes	see below

Color genesis block number	0, 3 or 4 bytes	unsigned 24- or 32-bit integer (small endian)
Color genesis txn byte offset in block	0, 3 or 4 bytes	unsigned 24- or 32-bit integer (small endian)
Prefix of genesis transaction ID	0 or 4 bytes	first 4 bytes of genesis transaction ID
Index of first input in this txn from which color is transferred	0, 1 or 2 bytes	unsigned 8- or 16-bit integer (small endian)
Number of inputs in this txn from which color is transferred	0, 1 or 2 bytes	unsigned 8- or 16-bit integer (small endian)
Input of first output in this txn to which color is transferred	0, 1 or 2 bytes	unsigned 8- or 16-bit integer (small endian)
Number of outputs in this txn to which color is transferred	0, 1 or 2 byte	unsigned 8- or 16-bit integer (small endian)
Number of color units transferred per output	0, 1, 2, 3, 4, 6 or 8 bytes	unsigned 8- to 64-bit integer (small endian)

The size of each field is determined by the packing metadata byte/s. The first metadata byte is interpreted bitwise as below. Bit strings are shown with the most significant bit on the left:

Bit or bits	**Related fields**	**Values and meanings**
128 + 64	Color genesis block number, txn offset in block and txid prefix	`00` = omit fields. If this is the first transfer, it is invalid, otherwise assume the same color genesis as the previous transfer. (In future having `00` in these bits of the first transfer can be used to indicate a different protocol version or packing structure. For now they would render the entire list of color transfers invalid.) `01` = 3 bytes for block, 3 bytes for txn offset, 4 bytes for txid. `10` = 3 bytes for block, 4 bytes for txn offset, 4 bytes for txid. `11` = 4 bytes for block, 4 bytes for txn offset, 4 bytes for txid.
32 + 16 + 8	Indices of first input and output, number of inputs and outputs	`000` = omit fields. If this is the first color transfer, transfer from input index 0 only to output index 0 only. Otherwise use the input and output ranges from the previous color transfer. `001` = omit fields. If this is the first color transfer, transfer from input index 0 only to output index 1 only. Otherwise use the input range from the previous transfer and the single output which is immediately after the output range from the previous transfer. `010` = omit fields. If this is the first color transfer, transfer from input index 0 only, otherwise use the input range from the previous transfer. Transfer to all transaction outputs in order. `011` = omit fields. If this is the first color transfer, transfer from input index 1 only to output index 0 only. Otherwise use the single input which is immediately after the input range from the previous transfer, and the same output range as the previous transfer.

		`100` = omit fields. If this is the first color transfer, transfer from all the inputs to output index 0 only. Otherwise transfer from all inputs to the output range from the previous color transfer. `101` = omit fields. If this is the first color transfer, transfer from all the inputs to output index 1 only. Otherwise transfer from all inputs to the single output which is immediately after the output range from the previous transfer. `110` = omit fields. Transfer from all inputs to all outputs in order. `111` = use second metadata byte to specify packing.
4 + 2 + 1	Number of color units transferred	`000` = omit field. If this is the first color transfer, assume a value of 1, otherwise take the value from the previous color transfer. `001` = use 8-bit unsigned integer (1 byte) `010` = use 16-bit unsigned integer (2 bytes) `011` = use 24-bit unsigned integer (3 bytes) `100` = use 32-bit unsigned integer (4 bytes) `101` = use 48-bit unsigned integer (6 bytes) `110` = use 64-bit unsigned integer (8 bytes) with max of 2^{63}-1. `111` = omit field and assume a value of 2^{63}-1, i.e. transfer all available units of color across

If the ‘input and output index ranges’ bits of the first packing metadata byte are `111`, then a second packing metadata byte is used to define how the input and output ranges are defined:

Bit or bits	**Related fields**	**Values and meanings**
128 + 64	Reserved	`00` (other values make this and subsequent transfers invalid)
32 + 16 + 8	Index of first input and number of inputs	`000` = omit fields. If this is the first color transfer, transfer from input index 0 only, otherwise use the input range from the previous transfer. `001` = omit fields. If this is the first color transfer, transfer from input index 1 only, otherwise use the single input which is immediately after the input range from the previous transfer. `010` = omit number of inputs and use 1 byte for the index of a single input. `011` = omit number of inputs and use 2 bytes for the index of a single input. `100` = 1 byte for first input, 1 byte for number of inputs. `101` = 2 bytes for first input, 1 byte for number of inputs. `110` = 2 bytes for first input, 2 bytes for number of inputs. `111` = omit fields. Transfer from all inputs.
4 + 2 + 1	Index of first output and number of outputs	Same as above, with ‘output’ substituted for ‘input’.

As discussed in the previous section, the goal of this encoding scheme is to enable a long list of color transfers to be specified within the byte restriction of a single transaction. For example, a specific amount of color could be broadcast from one input to all outputs using under 20 bytes.

For a color transfer involving a range of inputs, the color is drained from each input in numerical order, in order to transfer color to the output/s in the transfer. Each input is emptied in order before any color from the next input is used. For a color transfer involving a range of outputs, the specified amount of color is transferred to each output in numerical order. If there is insufficient color in the input/s to transfer the amount specified to every output, priority is given to earlier outputs, and as much as possible is transferred up to the amount specified.

Now let's examine the edge cases. For each color transfer, if no color genesis transaction matches the block number, transaction index and transaction ID prefix, then the color transfer is void and does not take place. If some of the input or output indices within the range specified in a color transfer are not present in the transaction, those inputs and outputs are skipped for the color transfer. (In theory we could also allow color to be transferred to the block miner as a transaction fee, i.e. to the output of the coinbase transaction of the block that successfully incorporates our transaction. This is a fascinating possibility since it allows non-bitcoin fees to be offered to miners.) If the packing metadata byte/s for a transfer are invalid, then the transfer does not take place, and any subsequent transfers in the list are ignored.

Color HTML metadata

Recall that each color genesis transaction contains a domain name, which is complemented in order to create a full URL for the color specification web page. When responding to a request for this URL, the web server should return a 404 error for URLs that are not recognized. This is crucial because the domain name of the color specification page will be presented clearly to the user of color-aware bitcoin clients, enabling them to assess whether a color called “Overstock credits” was genuinely issued by that company. (Clearly more thought will be required about how to prove the identity of a color issuer and prevent forgeries, but these issues must be solvable, since people are willing to use the web to perform important transactions despite the risk of phishing, homograph attacks, etc.... Some possibilities include: digital signatures using certificates issued by trusted bodies, additional DNS records in the style of the Sender Policy Framework for email, or services which monitor for unauthorized color issues using known brand names.)

The response for the request should include a human-readable web page about the color, along with some metadata within the `<HEAD>` section of the page. Redirections are permitted in order to provide a more user-friendly URL for bookmarking, but these should be done in Javascript rather than via HTTP 301 or 302 responses. This prevents color-aware bitcoin clients from worrying about edge cases such as redirection loops.

Each metadata field takes the following form:

```
<meta name=”coco:key” content=”value”>
```

The bold terms **`key`** and **`value`** are substituted according to the following table:

Key	Explanation	Example value

name	Name of the color for display	Overstock Credits
description (optional)	More information about the color	Allows you to get discounts from our store!
contract	Absolute URL of the contract	http://www.overstock.com/color-contract-1.pdf
multiple (optional)	Multiply the raw quantity of the asset by this number for display	0.01
format (optional)	How to display quantity	* credits (must contain one * which is substituted for the display value)
format_1 (optional)	How to display quantity if the display value is exactly 1	1 credit
color (optional)	HTML-style hexadecimal color for display	660000
icon (optional)	Absolute URL for icon to show for the color/asset (PNG only)	http://www.overstock.com/color-credits.png
feed (optional)	Absolute URL of RSS 2.0 feed for holders of the color	http://www.overstock.com/color-1.rss
tracker (one or more times)	URL of the tracking service for this color	http://colors.overstock.com/

To make life easy for wallet developers, all color specification web pages must use UTF-8 encoding. In addition, all references to external assets in the metadata should use absolute URLs, to avoid clients having to resolve relative URLs.

As mentioned earlier, the linked `contract` must be in a self-contained format such as PDF (without external assets), UTF-8 encoded plain text, JPEG or PNG. The file type will be determined by the color-aware wallet based on the URL suffix `.pdf`, `.txt`, `.jpg`/`.jpeg`, `.png` (i.e. wallets don't need to read the MIME type returned by the web server.) Wallets must explicitly prevent contracts being accepted if they are in HTML format, since HTML web pages almost always reference external assets such as images whose substitution could completely change the HTML's meaning.

The `exponent`, `format`, `format_1`, `color` and `icon` fields enable some control over how the quantities should be displayed in color wallets. Showing each asset in a different (but not too brash) color is a nice way to connect with the notion of bitcoin colors in the consumer's mind, and the color should match the icon, if provided. Icons should be square (which transparency permitted) and at least 32x32 pixels in size, which the wallet can scale as necessary. Note that there is also the quantity presentation specification inside the color genesis transaction, which fixes some aspects of the quantity display which must never be changed (for now, only the positive or negative power of 10 by which the number of units should be multiplied before being displayed to the user). The `feed` field enables notifications to be issued to color holders via RSS 2.0, for display in the appropriate fashion inside a color-aware wallet. Any critical notifications should also be posted to the color description page itself.

With the exception of `tracker`, each metadata field should only appear once. Some are optional, as indicated. In the event that a field appears more than one time, the first value should be taken as canonical. Multiple `tracker` fields enable redundancy to be provided for color tracking, so that if one server fails to respond, another can be queried. (We could also allow weights to be assigned to each tracking server, to manage the respective number of queries that are sent to each one.)

Color tracking queries

In order for a color-aware wallet to display bitcoin balances correctly, it is assumed that the wallet already knows the list of bitcoin transaction IDs and output indices with unspent outputs that it has the keys to unlock. Therefore all that is required is for the color tracking server to receive a list of (txid, output) tuples, along with a list of colors that the wallet is interested in and which the server tracks. It response, it returns the number of colors units for each combination of tuple and color.

Communication with color tracking servers uses JSON-RPC over HTTP, following the example set by the standard bitcoin software. The only method name currently supported is `gettxoutscolors` ("get transaction outputs' colors"). The query is sent in JSON format within the raw payload of an HTTP POST request to the URL provided in the `tracker` metadata field on the color specification page. The response is sent back in JSON format.

Example JSON-RPC request that requests the balance of two transaction outputs for two colors:

```
{
  "id": "any_request_id",
  "method": "gettxoutcolors",
  "params": {
    "txouts": [
      {
        "txid": "7f25b380f109b3b013e3f8c1ccd5a904b9bbf50bf23f98acb0b9c7586c415c79",
        "index": 3
      },
      {
        "txid": "c7ea20fa1e8b236b92c0f5a9d9e43e1bce9095faf81ae6bce636fe883b915645",
        "index": 2
      }
    ],
    "colors": [
      "f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9cea205b009ca73dd04470b9a6",
      "7fe0bbe5a3a37088b8a3065ed8b2ae1624e80c5a2a023568f197e4cce85310f3"
    ]
  }
}
```

Example response for the above request:

```
{
  "id": "any_request_id",
  "error": null,
  "result": {
    "f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9cea205b009ca73dd04470b9a6": [
      {
        "txid": "7f25b380f109b3b013e3f8c1ccd5a904b9bbf50bf23f98acb0b9c7586c415c79",
        "index": 3,
        "units": 146
      },
      {
        "txid": "c7ea20fa1e8b236b92c0f5a9d9e43e1bce9095faf81ae6bce636fe883b915645",
        "index": 2,
        "units": 0
      }
    ],
    "7fe0bbe5a3a37088b8a3065ed8b2ae1624e80c5a2a023568f197e4cce85310f3": [
      {
        "txid": "7f25b380f109b3b013e3f8c1ccd5a904b9bbf50bf23f98acb0b9c7586c415c79",
        "index": 3,
        "units": 146
      },
      {
        "txid": "c7ea20fa1e8b236b92c0f5a9d9e43e1bce9095faf81ae6bce636fe883b915645",
        "index": 2,
        "units": 0
      }
    ]
  }
}
```

(Better docs to follow…)

This gives the balance in units for each specified color for each specified transaction output, where the color is specified by its genesis transaction ID. If a color is not recognized and tracked by this server, the corresponding array will be `null`. If the output color quantities for a transaction ID are not yet known by the server, either because it has not seen the transaction yet, or because it has not seen one of its predecessors, then the corresponding `units` value will be `null`. (Better error reporting can be added later on,)

Color-aware wallet recommendations

- When a new color specification is seen for the first time in an incoming transaction, retrieve the appropriate genesis transaction from the blockchain. Immediately verify that the online color specification exists and cache the color name and linked contract locally. Verify these against the hash in the genesis transaction. Assuming everything checks out, prompt the user with something like this message:

  *You've been sent a new color called **Overstock Credits** from <**overstock.com**>. Click for <more information> or <the contract>. Would you like to track this color?*

Items in **bold** above are modified based on the color name and domain of the color specification URL respectively. Items in <angle brackets> are linked to the domain of the color specification, color specification page, and cached contract respectively.

- If the user chooses not to track a color, information about it should be stored somewhere, to enable users to 'recover' a color that they previously rejected.

- If the user chooses to track a color, its balance should be shown alongside the bitcoin balance, using the appropriate presentation. The name of the color should be clickable, to take the user to the color specification page. There should also be an easy way to view the cached contract for the color at any time.

- If a color specification is less than 7 days old, show some visual warning that indicates this. This allows users to be cautious with newly issued colors that might be forgeries (i.e. someone hacks into a website, create a color specification, then issues colors from it themselves.) The warning should appear both in the 'new color' message and continuously alongside the amount as it is displayed.

- Once every 24 hours, check that the color specification page is still available at the expected URL, and that the online name, description and contract matched the cached values (and, by implication, the hash in the genesis transaction). Alert the user if this is not the case, since it may indicate that an issuer is withdrawing a forgery or reneging on their promise. (Though simple server downtime in which there is no response should be dealt with silently by retrying at least once after a suitable delay such as an hour.)

- If a tracking server for a color cannot be queried successfully, there are several options: (1) attempt to calculate the quantity of the color by backtracking through transactions, (2) present the last known quantity of the color with some symbol (?) to denote that the sum cannot be verified, (3) show no value for the color whatsoever, (4) use some backup service that tracks every color that has been issued. Care should be taken to avoid panicking users in response to a little downtime of the tracking server. Note also that at the current bitcoin network capacity, there should be no difficulty in setting up a number of public servers that track all colors across all transactions. These could be used as a backup if an issuer's color server fails, and also to verify the responses from issuers' servers.

- When creating a payment to send elsewhere, allow users to send bitcoin and one or more colors simultaneously, to save on transaction fees.

- If you are sending some color to someone else via the last output, be explicit about that color transfer in the transaction information, even if this is not strictly necessary because all color flows to the last output by default. If the recipient has not seen the color before, this ensures that they are able to detect and start tracking it immediately.

- When providing a bitcoin address for receiving, display it in both representation formats – color-aware to receive colors, and in regular bitcoin style for backwards compatibility with bitcoin-only services.

Feedback is welcome, preferably in comments on the document or this thread.