Proprietary Metablock Chains for Arbitrary Data

Krona Rev

January 12, 2014

Abstract

We describe a method for creating chains of arbitrary data which are timestamped and secured by the Bitcoin network. Unlike other methods, the method described here does not destroy coins and does not create unspendable transaction outputs. Furthermore, the method does not add a significant amount of data to the Bitcoin block chain.

1 Introduction

The Bitcoin block chain provides a secure way of timestamping data [2]. The primary use of the block chain is to secure bitcoin transactions, but it can be used to timestamp, for example, hashes of arbitrary data. Once such a hash is in the block chain, it provides *proof of existence* of the data at that time. One way of doing this is to use the hash of the data to form a bitcoin address for which no private key is known. This has the unfortunate effect of destroying the coins spent to this address while also increasing the number of unspent transaction outputs. Since the set of unspent transaction outputs must be accessible in order to prevent double spending, having unnecessary unspent transaction outputs is a practical problem.

In this paper we describe a way to create chains of metablocks containing arbitrary data. The metablocks are certified using Bitcoin transactions without destroying bitcoins and without creating unspent transaction outputs. Each metablock includes a public key which can be used for two purposes: the public key modified by a hash of the metablock is used to certify the metablock and the public key is also used to determine who can certify a successor of the metablock. Since only someone with knowledge of the corresponding private key can certify a successor, we are describing a proprietary metablock chain. That is, there is always an owner who can decide what will be published in the next metablock. Ownership may be passed from the current owner to a new owner by publishing a public key from the new owner in the next metablock.

Metablocks can be published at the frequency desired by the proprietor (though the frequency of Bitcoin blocks places a hard limit). Infrequent publication of metablocks means placing less of a burden on the Bitcoin block chain. Each certifying Bitcoin transaction can be small, since it only requires a single input and a single output. In practice, this would mean publishing a metablock once a day would result in adding roughly 100 KB to the Bitcoin block chain in a year (estimating 1/4 KB per transaction). Assuming each of these transactions includes a miner fee of 0.1 mbits (millibits), this would cost the proprietor 36.5 mbits a year.

We briefly describe some ideas of how such a proprietary metablock chain could be used.

A notary service could maintain such a metablock chain in order to provide proof of existence. A similar service is provided by `www.proofofexistence.com`, but the outputs of the corresponding Bitcoin transactions created by this service are unspendable. Also, a metablock chain would allow someone the option of including the full data (not just the hash) in the metablock. This could be especially important if the data being notarized is a contract the parties wish to make public. Such a metablock chain could also be used to publish timestamped PGP public keys or associate ip addresses with domain names. A new metablock could be published either on demand (when someone wants something notarized) or published as a digest metablock on a regular basis (e.g., weekly).

A proprietor of a metablock chain could publish metablocks offering to sell crypto-financial instruments, such as options based on the bitcoin price. Since the contracts would be public, the history can be independently audited and the reputation of the proprietor of the metablock chain would depend on properly settling such contracts.

In Section 2 we give an informal description of metablock chains. In Section 3 we give an mathematical description of metablock chains and argue that certain desirable properties hold. In Section 4 we describe an example of a metablock chain, the first three metablocks of which have already been created and certified. In Section 5 we conclude.

2 Informal Description

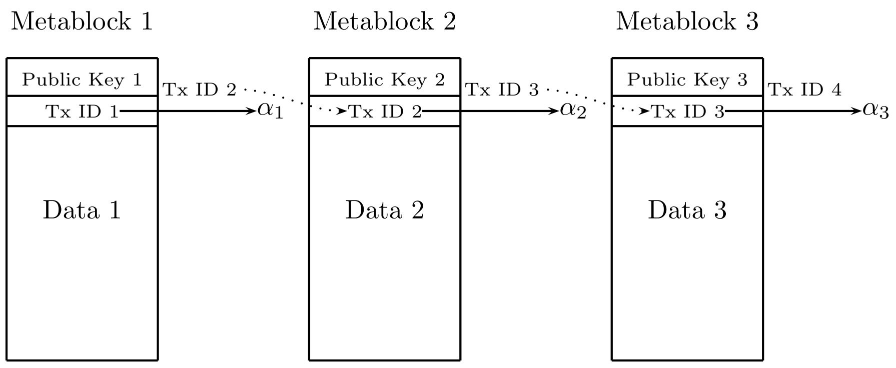
A metablock consists of a public key, a transaction id and some arbitrary data. Metablocks are certified by entering certain kinds of Bitcoin transactions into the Bitcoin block chain. This serves both to timestamp the metablock and gives a way to check the integrity of the data in the metablock. The public key indicates who is permitted to create certified successors of the metablock. The transaction id indicates a transaction from which we must spend in order to certify the metablock.

One idea to certify a metablock would be to hash the contents, and then use this hash to create an artificial bitcoin address. This technique is currently used sometimes to timestamp data. However, since no one knows the private key for this artificial bitcoin address, the effect is to make the bitcoins at this output unspendable.

Another idea is to use a hash of the contents as the private key for an address to which we spend some bitcoins. This would work in principle, but since everyone would be able to hash the contents and compute the private key, the owner would lose control of the coins.

We use a different idea. The metablock includes a public key for which we know the corresponding private key. A hash of the contents of the metablock is used to modify both the public key and private key to another key pair. Everyone can compute the modified public key from the metablock. (Indeed this is what allows arbitrary agents to verify that the metablock is certified by a Bitcoin transaction.) On the other hand, knowledge of the original private key is needed to compute the modified private key.

As shown in the figure below, we can begin with a metablock which includes a public key and a transaction id. From the public key and a hash of the metablock we can compute an address α_1. We certify the metablock by spending from the transaction with the transaction id to the address α_1. Since we know the private key for the original public key, we will also know the private key for α_1. In order to create a successor metablock, we simply include the transaction id for the transaction spending to α_1 as part of the second metablock. The second metablock also declares its own private key which will be used, along with a hash of the second metablock, to create an address α_2. We certify the second metablock by spending from the transaction certifying the first metablock to the address α_2. The id for this latest transaction is included in the third metablock and we certify the third metablock by spending from the transaction to an address computed from the third metablock. This can continue indefinitely.



More details about computing the addresses α_1, α_2 and α_3 and corresponding private keys are described in the next section. We will also argue in the next section why the certified metablocks form a chain.

3 Mathematical Description

Bitcoin uses elliptic curve cryptography with the particular parameters chosen as secp256k1 in [3]. One of these parameters is a base point (x_G, y_G) on the selected elliptic curve. The private keys are certain positive integers and the public keys are nonzero points on the curve. From a private key κ, the corresponding public key (x, y) is computed as $\kappa(x_G, y_G)$, where this scalar multiplication is defined as the iteration of the group addition on the curve. Bitcoin transactions spend to addresses α which are computed using hashes of public keys. We use the notation $\alpha(x, y)$ to refer to the address of a public key (x, y).

We will also consider certain offsets of public keys. Suppose m is an integer. If $m(x, y)$ is a nonzero point on the curve, then it is also a public key and there is a corresponding address $\alpha(m(x, y))$. Note that if κ is the private key for (x, y), then $m\kappa$ is the private key for $m(x, y)$.

We will use the hash of some information we wish to certify as offsets. We use $\sharp(\beta)$ to refer to the (SHA-256) hash of some data β.

The Bitcoin block chain collects blocks of valid Bitcoin transactions. These transactions are identified by transaction ids, which are also hashes.

A *certificate* is a pair (h, α) where h is a hash value (intended to be the transaction id for a transaction in the Bitcoin block chain) and α is an address.

Let τ be a Bitcoin transaction. We say τ *corresponds to certificate* (h, α) if input 0 of τ spends output 0 of the transaction with id h and output 0 of τ spends to address α. Clearly, a transaction can correspond to at most one certificate.

Not all transactions correspond to certificates. For example input 0 of a transaction might spend output 1 of another transaction. On the other hand, multiple potential transactions might correspond to a single certificate since we have not constrained other potential inputs and outputs of the transaction.

From now on, we restrict our attention to transactions confirmed in the Bitcoin block chain. Consequently, each certificate corresponds to at most one transaction. We say a certificate is *in the block chain* if a corresponding transaction is in the Bitcoin block chain. Note that if (h, α) and (h, α') are in the block chain, then $\alpha = \alpha'$ (since Bitcoin prevents double spending).

Only someone with knowledge of the private key for the address at output 0 of the transaction with id h can enter a certificate (h, α) into the block chain. Let us refer to someone with such knowledge as an *owner of* h. Similarly, we refer to someone with knowledge of the private key corresponding to the address α as an *owner of* α. Ownership of α is not required to enter a certificate (h, α) into the block chain.

A *metablock* β is a triple $((x, y), h, d)$ where (x, y) is a public key, h is a transaction id and d is a sequence of bytes.

Let β be a metablock $((x, y), h, d)$. We say β is *certified by* h' if h' is the transaction id for a transaction τ' in the Bitcoin block chain corresponding to the certificate $(h, \alpha(\sharp(\beta)(x, y)))$. We say β is *certified* if there is such an h'. Only an owner of h can certify a metablock $((x, y), h, d)$.

Let $\beta = ((x, y), h, d)$ and $\beta' = ((x', y'), h', d')$ be metablocks. We say β' is a *successor of* β and write $\beta \mapsto \beta'$ if β is certified by h'.

Suppose $\beta = ((x, y), h, d)$ is certified by h'. Successors are easy to forge: every metablock $((x', y'), h', d')$ is a successor. However, a successor metablock $\beta' = ((x', y'), h', d')$ can only be certified by entering a certificate

$$(h', \alpha(\sharp(\beta')(x', y')))$$

into the block chain. This can only be done by an owner of h'. Note that h' is the transaction id for a transaction τ' corresponding to the certificate $(h, \alpha(\sharp(\beta)(x, y)))$. This implies output 0 of τ' spends to address $\alpha(\sharp(\beta)(x, y))$. The owners of h' are (by the definition of owners of transaction ids above) the owners of $\alpha(\sharp(\beta)(x, y))$. Consequently, only an owner of $\alpha(\sharp(\beta)(x, y))$ can certify a metablock $((x', y'), h', d')$.

Let κ be the private key for (x, y), so that $(x, y) = \kappa(x_G, y_G)$. Since $\sharp(\beta)(x, y) = \sharp(\beta)\kappa(x_G, y_G)$, $\sharp(\beta)\kappa$ is the private key for $\sharp(\beta)(x, y)$. That is, from the private key κ for the public key (x, y) in a metablock and a hash $\sharp(\beta)$ of the metablock we can easily derive the private key needed to certify a successor of a metablock $((x, y), h, d)$. Essentially, each metablock gives a public key indicating who can certify a successor to the metablock. We refer to someone with knowledge of the private key for (x, y) as a *recipient of the metablock* $((x, y), h, d)$. By creating and certifying a successor $((x', y'), h', d')$ of β, the recipient of β is indicating the recipients of the successor by giving the public key (x', y'). In practice, this could be the same proprietor using a sequence of distinct public keys, or this could indicate passing the ownership to new proprietors.

We also need to know that each metablock has at most one certified successor. For this, we will argue that it is computationally infeasable to create more than one certified successor. Suppose $\beta = ((x, y), h, d)$, $\beta' = ((x', y'), h', d')$ and $\beta'' = ((x'', y''), h'', d'')$ are certified metablocks with $\beta \mapsto \beta'$ and $\beta \mapsto \beta''$. Since β' and β'' are both certified, the certificates $(h', \alpha(\sharp(\beta')(x', y')))$ and $(h'', \alpha(\sharp(\beta'')(x'', y'')))$ are both in the block chain. Since h' and h'' are transaction ids for transactions spending output 0 of the transaction referred to by h, h' must be the same as h''. Since the two certificates $(h', \alpha(\sharp(\beta')(x', y')))$ and $(h', \alpha(\sharp(\beta'')(x'', y'')))$ are in the block chain, we must have $\alpha(\sharp(\beta')(x', y')) = \alpha(\sharp(\beta'')(x'', y''))$. Note that $\sharp(\beta'')$ depends on (x'', y'') and d'', so that if either (x'', y'') differs from (x', y') or d'' differs from d', then $\sharp(\beta'')$ will differ from $\sharp(\beta')$ in a cryptographically unpredictable way. For this reason it seems infeasable to find different $\beta' = ((x', y'), h', d')$ and $\beta'' = ((x'', y''), h'', d'')$ so that the addresses $\alpha(\sharp(\beta')(x', y'))$ and $\alpha(\sharp(\beta'')(x'', y''))$ are the same.

A *metablock chain* is a finite sequence $\langle \beta_0, \ldots, \beta_n \rangle$ of certified metablocks such that $\beta_i \mapsto \beta_{i+1}$ for each $i \in \{0, \ldots, n-1\}$.

4 Example

We now describe an example of a metablock chain which currently includes three metablocks. The computations were done using an implementation of Hierarchical Deterministic Wallets (BIP0032) [5] available on github [4]. The three metablocks described here are also available at [4]. The transactions were created and sent by hand using the debug console of the Satoshi QT client.

Each metablock $\beta = ((x, y), h, d)$ is given as a text file. The first line of the text file is the uncompressed public key in hex format, and so is a hex number beginning with 04, continuing with the hex representation of x and ending with the hex representation of y. The second line is the transaction id h. The rest of the file contains the data d. The hash $\sharp(\beta)$ is given by computing the SHA-256 of the file (using `sha256sum`).

The first metablock includes an example of a contract which might be included in such a metablock. and the PGP public key of the author (Krona Rev). The second metablock includes a PGP signed message signed using the public key in the previous metablock. The third metablock includes an episode listing for the science fiction series Blake's 7. The contract in the first metablock identifies two parties via Bitcoin addresses and the contract is signed using the private keys corresponding to both addresses. The exact metablocks are available as files `MetaBlock1`, `MetaBlock2` and `MetaBlock3` at [4]. The full details about the public keys, private keys, and commands are in the `README.md` at [4].

The first metablock β_1 is $((x_1, y_1), h_1, d_1)$. The public key (x_1, y_1) is given in the first line as an uncompressed public key: a hex prefix of 04, x_1 as a 256-bit hex number (i.e., 64 hex characters) and y_1 as a 256-bit hex number (i.e., 64 hex characters). The exact line (separated into 3 lines here) is

```
04\
4E964BA401D7395B86251EC3E924D6A20CA8805164E36FBEBA5627A8848C39B3\
7842C2E7D512D027F849F4BE3662B5A0657583AD16228D7690C9BF5B2AE62BAB
```

The corresponding private key κ_1 (in base 58 bitcoin "wallet import format") [1] for this public key is

```
5HsoiMv5TH6asaXhgxSZAkp5pVSMp1QwdWRMiKGGN6Lhe12Hdgj
```

We will need to use this private key to certify the first metablock. The hash h_1 is

```
925e591d7873c1a59ada5a7b3dd80f90b7f093b45059e0b780640f15cafba886
```

Output 0 of this transaction spends some mbits to some address which is not relevant here. The data d_1 is specified by the rest of the metablock after the first two lines. The SHA-256 hash $\sharp(\beta_1)$ of the metablock is

```
ee834e4747d3327201acd2b8158d36a758d4076fca6c4778f8f004107c43ad9b
```

We use this value to offset the public and private keys above. For example, the modified private key $\sharp(\beta_1)\kappa_1$ is

```
./bip0032sbclexec offsetpriv ee83...9b 5Hsoi...Hdgj
5KcXkCerDYyEgZpHHu8ZuvkgGW5bgKzNUyPz8jCnDeQ1TuUdVWQ
```

The modified public key $\sharp(\beta_1)(x_1, y_1)$ and address can be obtained as follows:

```
./bip0032sbclexec offsetpub ee83...9b 044E96...AB
Modified Public Key: 04\
E852E940DCD5E1A213D767ED76085D955BC0F8FA2DBAD6588589F736B333FEBB\
DFF2637BEC1195ACF43F397DDAA11D4E3401D97626554B86B29D34EA51AE4C08
Corresponding BTC Address: 1ECYmKcntwjkEJf8bD1WEyKFzLBigxBEZ5
```

In order to certify the first metablock, we create a transaction where input 0 spends output 0 of h_1 and output 0 spends to the address $\alpha(\sharp(\beta_1)(x_1, y_1))$ computed above:

```
1ECYmKcntwjkEJf8bD1WEyKFzLBigxBEZ5
```

We sign and send the transaction and obtain the corresponding id h_2:

```
4b5066707134d715f7b9d2efe3d9f7e0d320689787611ed7cbe1d5cbf98c5768
```

While this process could be fully automated, it currently is not. However, one need not go through all the steps above explicitly. Instead, the implementation will describe the process in some detail if one simply calls the following command.

```
./bip0032sbclexec certifymetablock MetaBlock1 ...someprivatekey...
```

To verify that h_2 certifies β_1, anyone can compute the address $\alpha(\sharp(\beta_1)(x_1, y_1))$ from β_1 and verify that input 0 of h_2 spends output 0 of h_1 and output 0 of h_2 spends to $\alpha(\sharp(\beta_1)(x_1, y_1))$.

The second metablock β_2 is $((x_2, y_2), h_2, d_2)$ where h_2 is the transaction id obtained above. Using h_2 in β_2 ensures β_2 is a successor of β_1. We do not give the details about the public key (x_2, y_2) and corresponding private key κ_2. The details are available in at [4]. We obtain an address $\alpha(\sharp(\beta_2)(x_2, y_2))$ by modifying the public key using the hash of the metablock as above. We certify β_2 by creating a transaction for which input 0 spends output 0 of h_2 and output 0 spends to the address above. We sign this transaction with the modified private key κ_1 above. The details for the process can be obtained by calling

```
./bip0032sbclexec certifynextmetablock MetaBlock1 MetaBlock2\
                                   5Hso...Hdgj
```

where `5Hso...Hdgj` is the unmodified private key κ_1. This command hashes the first metablock in order to compute the modified private key $\sharp(\beta_1)\kappa_1$ for the new input, hashes the second metablock in order to compute the modified address $\alpha(\sharp(\beta_2)(x_2, y_2))$ for the new output, indicates how to create the transaction spending from output 0 of h_2 (signing with $\sharp(\beta_1)\kappa_1$) and spending to $\alpha(\sharp(\beta_2)(x_2, y_2))$ with a new output 0. The transaction id for this transaction is h_3:

```
97e023eddaf5bb867c5caf25bfd5b02fc1ce2588a2077cb22bc9a440d5f0e0f2
```

The third metablock $\beta_3 = ((x_3, y_3), h_3, d_3)$ is a successor to β_2 since h_3 certifies β_2. In this case, the public key (x_3, y_3) is published, but the private key is not. Only someone with knowledge of the corresponding private key can create a successor to β_3. Again, we certify β_3 by spending from output 0 of h_3 as input 0 to $\alpha(\sharp(\beta_3)(x_3, y_3))$ as output 0 of the new transaction. The transaction id for the transaction which certifies β_3 is

```
f2fb9b6481ed928aedef27279f7ad3d8db657c81407eee20d5f3a21daf15a28d
```

Hence we have three certified metablocks with

$$\beta_1 \mapsto \beta_2 \mapsto \beta_3$$

The implemenation can also be used to verify that a certain metablock is certified by a transaction or that a certain metablock is a successor of another metablock. Here are two examples (where some details are elided):

```
./bip0032sbclexec verifymetablocksucc MetaBlock1 MetaBlock2

To verify the metablock successor get transaction
4b5066707134d715f7b9d2efe3d9f7e0d320689787611ed7cbe1d5cbf98c5768
and check that two properties hold:
input 0 spends output 0 of transaction 925e...86
and output 0 of spends to address 1ECYmKc...EZ5
```

```
./bip0032sbclexec verifymetablock MetaBlock3 f2fb...8d

To verify the metablock get transaction
f2fb9b6481ed928aedef27279f7ad3d8db657c81407eee20d5f3a21daf15a28d
and check that two properties hold:
input 0 spends output 0 of transaction 97e0...f2
and output 0 of spends to address 1BkwaK5...EPw
```

5 Conclusion

We have have defined the notion of a metablock chain which allows a proprietor to publish a sequence of metablocks containing arbitrary data in a way secured by the Bitcoin block chain. The method is inexpensive in that it only requires a small Bitcoin transaction to certify each metablock. It also does not destroy bitcoins and does not increase the number of unspent transactions.

Acknowledgements: Thanks to Satoshi Nakamoto, who has freed more slaves than anyone in history.

References

[1] Wallet import format, 2013. Available on the bitcoin wiki at `https://en.bitcoin.it/wiki/Wallet_import_format`.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[3] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, 2000. Version 1.0.

[4] Krona Rev. bip0032sbcl, 2014. `https://github.com/kronarev/bip0032sbcl`.

[5] Pieter Wuille. Hierarchical deterministic wallets, 2012. BIP0032 `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`.