Journal of Grid Computing manuscript No.
(will be inserted by the editor)

A journey into Bitcoin metadata

Massimo Bartoletti · Bryn Bellomy · Livio Pompianu

Received: date / Accepted: date

Abstract Besides recording transfers of currency, the Bitcoin blockchain is being used to save *metadata* — i.e. arbitrary pieces of data which do not affect transfers of bitcoins. This can be done by using different techniques, and for different purposes. For instance, a growing number of protocols embed metadata in the blockchain to certify and transfer the ownership of a variety of assets beyond cryptocurrency. A point of debate in the Bitcoin community is whether metadata negatively impact on the effectiveness of Bitcoin with respect to its primary function. This paper is a systematic analysis of the usage of Bitcoin metadata over the years. We discuss all the known techniques to embed metadata in the Bitcoin blockchain; we then extract metadata, and analyse them from different angles.

Keywords Bitcoin, blockchain, measurements

1 Introduction

The last few years have witnessed an increasing interest in Bitcoin, the first and most widespread decentralised cryptocurrency [40,48]. Bitcoin records currency transactions in a public, append-only data structure — the so-called *blockchain*. The blockchain is maintained by a peer-to-peer network, following a consensus protocol which ensures that tampering with the past transactions is computationally unfeasible [42].

M. Bartoletti
Università degli Studi di Cagliari, Dipartimento di Matematica e Informatica, Via Ospedale, 72, 09124 Cagliari, Italy
E-mail: bart@unica.it

B. Bellomy
ConsenSys, 49 Bogart St., Brooklyn, NY 11206
E-mail: bryn.bellomy@consensys.net

L. Pompianu
Università degli Studi di Cagliari, Dipartimento di Matematica e Informatica
E-mail: livio.pompianu@unica.it

The immutability of the Bitcoin blockchain, together with its openness, have inspired the development of new applications, that — going beyond transfers of currency — certify the existence of documents [16,21,25], track the ownership of assets [12,18,19], run smart contracts [14,29,36] or perform other useful tasks. These applications exploit Bitcoin transactions to "piggy-back" their *metadata*, i.e., pieces of data which are not inherent to currency transfers, but are needed to implement their application logic.

A debate about scalability has been taking place in the Bitcoin community over the last few years [1,22,23]. In particular, users argue over whether the blockchain should allow for storing these spurious data. Besides quantifying the impact of metadata on the effectiveness of Bitcoin, many other relevant aspects on the usage of metadata are still worth of investigation.

This paper is a systematic survey on the usage of metadata in Bitcoin, based on the analysis of the first 480,000 blocks in the blockchain, i.e. ∼245K transactions collected until 2017/08/10. Our main contributions can be summarised as follows:

1. We survey the existing techniques for embedding metadata in the Bitcoin blockchain, identifying 11 distinct ones. We compare these techniques, discussing their side effects, and we compare their evolution over time, quantifying the amount of metadata embedded through them.
2. We search the blockchain for metadata, and we parse them to infer the intended usage. To this purpose, we consider both metadata as single units of information, and as aggregates of pieces scattered through the blockchain (e.g., images). Overall, we recognise 7

different types of metadata. We quantify the amount and size of metadata by type.
3. We identify 45 distinct protocols which are used by applications to embed metadata in the blockchain. We classify them according to their application domain, and we measure the amount of metadata they produced. We analyse the correlation between embedding techniques, metadata types and protocols.
4. We compare the size of the extracted metadata with the overall size of the blockchain, and we investigate peaks of metadata that occurred over the years.
5. We make available a public dataset of metadata extracted from the blockchain [35], as well as the tools we have developed for our analyses [33, 38].

Structure of the paper. Section 2 is a minimalistic introduction to Bitcoin, containing all the technical background needed in the subsequent sections. In Section 3 we show and compare the techniques to embed metadata in the blockchain, presenting several statistics on their usage. In Section 4 we illustrate some techniques to parse metadata and reconstruct the original content; then, we categorize and quantify the metadata extracted from the blockchain. In Section 5 we investigate the protocols which embed metadata, we classify them, and quantify the volume of metadata produced by each protocol. Section 6 discusses how the usage of metadata impacts on the Bitcoin blockchain. Finally, Section 7 discusses some related works, and Section 8 draws some conclusions.

2 Background on Bitcoin

Bitcoin [48] is a decentralized infrastructure to exchange virtual currency — the *bitcoins*. Users interact with Bitcoin through *addresses*, by publishing *transactions* that transfer bitcoins from one address to another. The log of all transactions is recorded on the blockchain, a public and immutable data structure maintained by the nodes of the Bitcoin network. A subset of nodes, called *miners*, gather the transactions sent by users, aggregate them in blocks, and try to append these blocks to the blockchain. A consensus protocol based on moderately-hard "proof-of-work" puzzles is used to resolve conflicts that may happen when different miners concurrently try to extend the blockchain, or when some miner attempts to append a block with invalid transactions. Ideally, the blockchain is globally agreed upon, and free from invalid transactions, unless the adversary controls the majority of the computational power of the network [31, 42, 44]. The security of the consensus protocol relies on the assumption that miners are rational, i.e. that following the protocol is more convenient than trying to attack it. To make this assumption hold, miners receive some economic incentives for performing the time-consuming computations required by the protocol. Part of these incentives is given by the *fees* paid by users upon each transaction.

2.1 Transactions

To illustrate how transfers of bitcoins work, we consider two transactions T_0 and T_1, which we represent graphically as follows:

T_0	T_1
`previous transaction`: $\cdots$ `in-script`: $\cdots$	`previous transaction`: T_0 `in-script`: $sig_k(\bullet)$
`value`: v_0 `out-script`(T,σ): $ver_k(T,\sigma)$	`value`: v_1 `out-script`: $\cdots$

The transaction T_0 contains v_0 Satoshis (1 bitcoin = 10^8 Satoshis). A user can redeem this amount by publishing a transaction (e.g., T_1), whose `previous transaction` field contains the identifier of T_0 (displayed just as T_0 in the figure), and whose `in-script` field makes the `out-script`[1] of T_0 evaluate to true. When this happens, the value of T_0 is transferred to the new transaction T_1, and T_0 becomes unredeemable. A subsequent transaction can then redeem T_1 likewise. In the transaction T_0 above, `out-script` checks that σ is a valid signature (made with the key k) of the redeeming transaction. We denote with $ver_k(T,\sigma)$ the signature verification function, and with $sig_k(\bullet)$ the signature of the enclosing transaction, including all the parts of the transaction but its `in-script` (obviously, because it contains the signature itself).

Now, assume that T_0 is redeemable on the blockchain when someone tries to append T_1. This is possible if $v_1 \leq v_0$, and the `out-script` of T_0, applied to T_1 and to the signature $sig_k(\bullet)$, evaluates true.

The previous example shows the simple case of transactions with only one input and one output. In general, transactions have the form displayed in Figure 1. First, there can be multiple inputs and outputs (denoted with array notation in the figure): `in-counter` specifies the number of inputs, and `out-counter` that of outputs. Each input (resp. output) has its own `in-script` (resp. `out-script`). The two fields `in-script length` and `out-script length` denote the size of `in-script` and `out-script`, respectively. Since each output can be redeemed independently, `previous transaction` fields

[1] The fields `in-script` and `out-script` are called, respectively, `scriptPubKey` and `scriptSig` in the Bitcoin wiki.

T
version_no: k
in-counter: n
previous transaction[0]: T_0 previous out-index[0]: i_0 in-script length[0]: $\cdots$ in-script[0]: $\cdots$ sequence_no[0]: $\cdots$
$\vdots$
out-counter: m
value[0]: v_0 out-script length[0]: $\cdots$ out-script[0]: $\cdots$
$\vdots$
lock_time: s

Fig. 1: A Bitcoin transaction.

must specify which one they are redeeming (in the figure, `previous out-index`). A transaction with multiple inputs redeems *all* the (outputs of) transactions in its `previous transaction` fields, by providing a suitable `in-script` for each of them. The `lock_time` field specifies the earliest moment in time when the transaction can appear on the blockchain. The `version_no` field is currently set to 1. Transaction inputs contain also a 4-bytes field called `sequence_no`. Normally its value is `0xFFFFFFFF`, and it is ignored unless the transaction `lock_time` is greater than 0 [11].

In order for T to be appended to the blockchain, a few conditions must be satisfied: for instance, for each $k < n$, the `out-script` of the i_k-th output of the transaction T_k must evaluate to true when fed with the value in `in-script[k]`; further, none of the inputs must have been redeemed yet, and the sum of the values of all the redeemed outputs must be greater than or equal to the sum of the values of all outputs in T (see [30] for a formal specification).

The *Unspent Transaction Output set* (in short, UTXO set) is the set of redeemable outputs of all transactions in the blockchain.

2.2 Scripts

Bitcoin scripts [10, 11] are programs in a stack-based language featuring a limited set of logic, arithmetic, and cryptographic operators (but without loops). In the rest of this section we illustrate the pairs of `in-script` and `out-script` which are considered *standard* by the Bitcoin network [39]. In Section 3 we will then show how these scripts are commonly used for embedding metadata.

We use the typewriter font for denoting opcodes (e.g., `OP_CHECKSIG`), and italic for denoting bitstrings (e.g., *sig*). In Bitcoin scripts, these bitstrings are always preceded by a suitable `OP_PUSHDATA` opcode, which pushes the bitstring onto the stack. For the sake of simplicity, hereafter we omit these `OP_PUSHDATA` opcodes.

2.2.1 Pay to public key (P2PK)

```
# pay-to-Pubkey (P2PK)
in-script : sig
out-script: pubKey OP_CHECKSIG
```

This pair of scripts implements the signature verification function outlined in Section 2.1. The evaluation of the output script starts with *sig* on top of the stack, and then proceeds by pushing also *pubKey*. The opcode `OP_CHECKSIG` performs the signature verification, popping the two top elements of the stack, and evaluating to true if the verification succeeds.

2.2.2 Pay to public key hash (P2PKH)

```
# pay-to-PubkeyHash (P2PKH)
in-script : sig pubKey
out-script: OP_DUP OP_HASH160 pubKeyHash
            OP_EQUALVERIFY OP_CHECKSIG
```

This pair of scripts performs signature verification, similarly to the previous pair. The main difference with respect to P2PK is that the output script now contains the double *hash* of a public key, rather than the public key itself.

More in detail, the `in-script` contains a signature *sig* and a public key *pubKey*. The evaluation of the output script starts with *sig* and *pubKey* on the stack. The opcode `OP_DUP` duplicates the top element of the stack (i.e., *pubKey*), and `OP_HASH160` replaces the top element with its double hash. Then, *pubKeyHash* (the double hash of a key) is pushed into the stack. The opcode `OP_EQUALVERIFY` checks if the two top elements of the stack are equal: if so, they are popped, otherwise the script fails. Finally, `OP_CHECKSIG` performs the signature verification.

2.2.3 Multi-signature

```
# multi-signature
in-script: OP_0 sig1 ... sigM
out-script: M pubKey1 ... pubKeyN N
            OP_CHECKMULTISIG
```

This pair of scripts performs a multi-signature verification: the output can be redeemed if the `in-script` provides M signatures verified against N public keys, where $M \leq N$. The opcode `OP_CHECKMULTISIG` tries to verify the last signature with the last public key. If they match, it proceeds to verify the previous signature in the sequence, otherwise it tries to verify the signature with the previous key. Notably, `OP_CHECKMULTISIG` uses each key only once, therefore the order of the signatures in the `in-script` matters.

2.2.4 Pay to script hash (P2SH)

```
# pay-to-ScriptHash (P2SH)
in-script : v1 ... vN bitstring
out-script: OP_HASH160 hash OP_EQUAL
```

In this pair, firstly the `out-script` is evaluated with *bitstring* on top of the stack. The script checks that the hash of *bitstring* is equal to *hash*. If so, the script obtained by interpreting *bitstring* as a sequence of opcodes is executed (on the parameters *v1* ... *vN*). Summing up, the output can be redeemed if the `in-script` of the redeeming transaction provides a script whose hash coincides with the *hash* contained in `out-script`, and whose evaluation (on the parameters *v1* ... *vN*) yields true.

2.2.5 `OP_RETURN`

```
# op_return
out-script: OP_RETURN bitstring
```

An `out-script` containing `OP_RETURN` always evaluates to false, regardless of the value of the *bitstring*. Therefore the corresponding output is unspendable, and it can be safely removed from the UTXO set.

Currently, standard transactions can have only one occurrence of `OP_RETURN`: more precisely, if a transaction has more than one `out-script` with `OP_RETURN`, or an `out-script` with more than one `OP_RETURN`, or an `OP_RETURN` with more than one `OP_PUSHDATA`, then the transaction is not standard.

3 Embedding metadata in the blockchain

In Sections 3.1 to 3.7 we illustrate various techniques (as far as we know, all those used in practice) to embed metadata in the blockchain. In Section 3.8 we show how to split large pieces of metadata into smaller pieces that can be distributed among sets of transactions. In Section 3.9 we discuss how to commit to specific values without explicitly writing them in the blockchain. Section 3.10 gives some statistics on embedding techniques.

3.1 Value field

Transaction outputs specify the amount of Satoshis to send through the `value` field, of 8 bytes size. A first way to encode a message m in the blockchain is to build a transaction with an output whose value is the number that represents m. For instance, the BitcoinTimestamp protocol (see Table 5) exploits this method for saving a SHA256 hash. The hash is first split into 16 pieces, which are then translated into amounts of Satoshis. Finally, the protocol builds a transaction containing an output for each amount (e.g. see rows 1-2 of Table 1).

Although users can easily recover the moved funds (e.g. by specifying their own address as receiver), the disadvantage of this technique is that it requires to own at least the amount of Satoshis needed to represent m.

3.2 Input sequence

Some users exploited the 4 bytes in the `sequence_no` field for appending their own metadata. Although this technique does not have negative effects on the Bitcoin system, as far as we know no protocols use this technique (as shown in Table 5). We conjecture that protocols rely on other techniques because 4 bytes are not enough for implementing any relevant use case (see Section 4 for a description of protocols and use cases).

3.3 Pay to public key / Pay to public key hash

In P2PK (Section 2.2.1) the output script specifies the recipient of some bitcoins through a public key, encoded in 65 bytes (or 33 bytes when the key is compressed). Similarly, P2PKH (Section 2.2.2) uses the hash of the key, which is encoded in 20 bytes.

Users can embed an arbitrary message m (of suitable size) in the P2PK or P2PKH script of some output of a transaction T, by writing m in place of the expected key or hash. Assuming that Bitcoin uses secure signatures and collision-resistant hash functions, it is computationally unfeasible, in general, to craft a transaction which redeems such output of T.

A downside of this approach is that these unspendable outputs are indistinguishable from the spendable ones: actually, a Bitcoin node has no way of knowing whether or not a user exists who possesses the needed hash preimage (nor does it know that the data was never intended to represent an address in the first place).

#	Technique	Transaction identifier / Bitcoin address
1	Value	f6f89da0b22ca49233197e072a39554147b55755be0c7cdf139ad33cc973ec46
2		49a130ce4255fc91061c3d1170cbc256f51ed671256df837500d59183cfdd64f
3	OP_RETURN	d84f8cf06829c7202038731e5444411adc63a6d4cbf8d4361b86698abad3a68a
4	Vanity Address	1ponziUjuCVdB167ZmTWH48AURW1vE64q
5		1CounterpartyXXXXXXXXXXXXXXXUWLpVr
6		72162e9224dbadefb84834046ee8b4706af77f57fa4e8fd5aaf3255abf516807
7		1WeRe3jh9XiaAyabyiE2Mz4v8bbcB52Gy
8		1FineoW99TYAAZuRSkbZLrx65iTXELqHhv
9		18chaNzLXvAbYvkad7MH2LNrQmBzeXbWLo
10		1PoStJBYu49Ezqcwh1VeMWZgRopwcYwksY
11		1FAke1neYErMQLebVPYBAToTLvafr5ZPF6
12	Coinbase	4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
13	P2PKH	5970ae129d1141663bd5e441a1555c16fb1c0586dd05f40c1db3d3e81218ee41
14	P2SH	1e47936f37e71b98e8bafe51ddc902d59c1318bc556329ba4ab1996981785292

Table 1: Some transactions and addresses containing metadata.

As a result, the nodes of the Bitcoin network must keep these transactions in their UTXO set indefinitely. Since the UTXO set is usually stored in RAM for efficiency concerns [52], the bloating of the UTXO set negatively affects the memory consumption of nodes [32].

3.4 Pay to script hash

By using P2SH scripts (Section 2.2.4), metadata can be embedded in various ways. Similarly to the P2PKH technique (Section 3.3), one can embed metadata in the output script, in place of the *hash*. An alternative technique is to embed metadata in the input script, pushing them onto the stack with the OP_PUSHDATA instruction, and immediately afterwards removing them with OP_DROP. As long as the script completes its execution successfully and there is some nonzero value on the stack after completion, the transaction is valid [10]: indeed, there is no rule specifying that the data accumulated on the stack during the script execution must be cleared. Stack items that are below the topmost item at the end of execution are simply ignored. For instance, consider the following scripts:

```
in-script  : OP_PUSHDATA1 8
             0xaabbccddeeff0011 sig pubKey
             v1 ... vN bitstring
out-script: OP_HASH160 hash OP_EQUAL
```

First, the `in-script` and the `out-script` are concatenated; the result is a common P2SH script preceded by an OP_PUSHDATA1. The evaluation starts by pushing `0xaabbccddeeff0011` onto the stack. This is done through OP_PUSHDATA1, where the trailing 1 indicates that the next 1 byte contains the number of bytes to be pushed onto the stack (in the snippet above, 8 bytes). At the end of the execution, the top item on the stack is `true`, resulting from the OP_EQUAL. Underneath this value there is the metadata `0xaabbccddeeff0011`.

Note that transactions that make use of ignored OP_PUSHDATA for embedding metadata do not bloat the UTXO set: indeed, their outputs can be spent by valid addresses, because they do not need to overwrite the address fields in the transaction scripts. The work [26] provides further details on the P2SH technique.

3.5 OP_RETURN

Standard OP_RETURN transactions allows to store up to 80 bytes of arbitrary data (see e.g. row 3 of Table 1). An `out-script` containing OP_RETURN always evaluates to false, hence the output is unspendable, and its transaction can be safely removed from the UTXO set. In this way, OP_RETURN overcomes the UTXO consumption issue highlighted in Section 3.3.

3.6 Vanity address

Bitcoin addresses are the hash of ECDSA public keys. By a brute force search during the generation of the key pair, it is possible to obtain a so-called *vanity address*, where a few bytes are equal to a given string (see e.g. row 4 of Table 1)[2]. One can embed these bytes as metadata in transactions, e.g. by using the vanity address in P2PK scripts.

Although, in theory, the maximum size of the metadata corresponds to the size of an address (20 bytes), in practice this technique is practical only for metadata of the size of a few bytes. Longer metadata can be distributed among different vanity addresses; for instance, the transaction at row 6 of Table 1 transfers bitcoins from 5 vanity addresses (displayed at rows 7-11). By concatenating the first four characters (but the leading

[2] Open-source tools like Vanitygen generate vanity addresses with user-defined patterns.

1) of these addresses, we read the plain English words: *"We're fine, 8chan post fake"*.

A different use of vanity addresses is when *all* the bytes of the address are fixed, like e.g. at row 5 of Table 1. In this case, it is plausible that the corresponding private key is not known to anybody, so a P2PK output using such address cannot be spent. This fact is exploited to implement "Proof-of-Burn" protocols on top of Bitcoin (like e.g. in Counterparty): in these protocols, users receive some tokens in exchange for sending some bitcoins to an unspendable address.

3.7 Coinbase transaction

Miners specify how to redeem the reward for the mined block (and the fees of its transactions) through the first transaction of the block. This transaction does not have an input script, and it contains a field called `coinbase`, that miners usually fill in with metadata. The coinbase data size is between 2 and 100 bytes. Nevertheless, after block 227,835 the available space is reduced, since the Bitcoin Improvement Proposal 34 (BIP 0034) [27] requires the first bytes of the coinbase field to store the block height index.

Usually, the `coinbase` field is used by miners for identifying the mining pool, or for voting BIPs (for instance, when they voted to support either the BIP 0016 or BIP 0017 [28]). The most famous message embedded by using this technique is included in the genesis block (see e.g. row 12 of Table 1): *"The Times 03/Jan/ 2009 Chancellor on brink of second bailout for banks"*.

3.8 Distributing metadata

The techniques discussed in Sections 3.1 to 3.7 embed metadata within a single field of a single transaction. Below we describe three techniques that allow to split metadata among multiple fields and transactions.

Multisignature. This technique uses the multisignature script introduced in Section 2.2.3. Given $N-1$ pieces of metadata *v2*, ..., *vN*, the scripts are the following:

```
in-script:  OP_0  sig1
out-script: 1 pubKey1 v2 ... vN N
            OP_CHECKMULTISIG
```

This implements a 1-of-N multisignature: namely, only one signature (*sig1*) needs to be verified against some of the N public keys (*pubKey1*). The other "public keys" in the `out-script` — actually, the $N-1$ pieces of metadata — are irrelevant for the execution of the script.

Note that this technique bloats the UTXO set only until the transaction is redeemed.

Multiple inputs / outputs. Although Bitcoin imposes a hard limit of 10K bytes on the size of a single script [9], there is no limit to the number of inputs or outputs a transaction may contain. Hence, metadata bigger than 10K bytes can be split into smaller chunks, and distributed among many inputs or outputs within a single transaction. To ensure that the original data can be reconstructed from these fragments, one needs to fix an encoding. The simplest one is to store the chunks of metadata sequentially in N output scripts. Another common encoding, used e.g. by the BIT-COMM protocol, uses the amount of bitcoins transferred by each output to order the chunks (see e.g. row 13 of Table 1). Transactions using this technique may or may not bloat the UTXO set — that is determined by the structure of the individual transaction outputs.

Transaction chains. The previous techniques store metadata within a single transaction. However, this is not always ideal, or even possible. For instance:

- If the size of metadata exceeds the maximum block size, the transaction containing the metadata will be rejected by the network.[3]
- Large transactions require large fees. Even though, in theory, one can send a transaction with zero fee, in practice a transaction with no fee (or a low fee) is unlikely to be mined. Depending on current fee market dynamics, it may be more cost-effective to split the metadata across multiple transactions.
- Transactions greater than a certain size, or which contain more than one `OP_RETURN`, are considered non-standard, with the consequence that most nodes refuse to relay them. This limit has varied over time, and is now replaced by the concept of "transaction weight" which is similar, but accounts for Segregated Witness data in a different manner.[4]

Due to these considerations, metadata are often split into sets of transactions. A common technique is to connect transactions containing related data in a chain structure. When building a transaction chain, one of the techniques from Sections 3.1 to 3.7 is chosen for encoding data into each individual transaction. Then, a spendable transaction output is added to each transaction, to be redeemed by the subsequent transaction in the chain. The output must be spendable by an address over which the user embedding the data has control.

[3] See github.com/bitcoin/.../src/main.cpp#L829.
[4] See github.com/bitcoin/bitcoin/.../src/main.h#L56, and github.com/bitcoin/bitcoin/.../src/main.cpp#L644-L648.

3.9 Embedding *vs.* committing metadata

Several Bitcoin-based protocols notarize documents by embedding their hash on the blockchain. There are alternative techniques, e.g. `pay-to-contract` [43] and `sign-to-contract`[5], which allow one to commit to a hash without actually embedding it into a transaction. For instance, Btproof applies RIPE160 to the hash, to obtain a Bitcoin address; similarly, Originstamp daily aggregates hashes into a seed, hashes it into a private key, and then derives from it the corresponding public key and address. Both protocols pay a small fee to the generated address in order to publish it in the blockchain. ContractHashTool exploits elliptic curves for building an `in-script` that cryptographically commits to a given hash, without embedding the hash itself. Unless a protocol keeps a public track of its transactions (like, e.g., Originstamp), these techniques prevent external observers from inferring any metadata.

3.10 Statistics on embedding techniques

Table 2 shows the amount of metadata embedded with each technique. The leftmost column groups the techniques in three categories: the *Single* category contains the techniques which embed the whole piece of metadata into a single chunk; *Multi* contains the techniques which split an element into multiple chunks, but store all the pieces into a single transaction; finally, *Chains* gathers the techniques which spread pieces across multiple transactions. The third and fourth columns show the size (in bytes) of the field storing metadata, and where the field is located. The fifth column lists the techniques bloating the UTXO. The sixth column displays the date in which first chunk of metadata appears, the seventh one shows the number of times a technique has been used[6], and last two columns show the total and average size of metadata embedded.

In the first 480,000 blocks we count 4,582,661 chunks of metadata, for a total size of ∼ 99MB. Note that, since the chunks of the *Chains* type are a subset of those in the other categories, and the chunks in *Multi* are a subset of those in *Single*, to avoid counting the same piece of metadata multiple times, the totals only consider the values of the *Single* type. This number of chunks is a good indicator of the total number of transactions with metadata, since the fraction of transactions produced by *Multi* techniques is negligible. We observe that ∼75% of the space used by metadata is due to `OP_RETURN` transactions. The average size of transactions chains is higher than other methods, since this technique is used for embedding images and archives.

[5] See bitcointalk.org/index.php?topic=915828.msg10056796

[6] Elements of type *Single* are chunks; *Multi* elements are transactions; *Chains* elements are chains of transactions.

4 Analysis of Bitcoin metadata

In this section we present our techniques to parse metadata and reconstruct the original content. Then, we categorize the reconstructed items, and we measure them.

4.1 Collecting metadata

One of the most effective techniques for recognising chunks of metadata is to search strings for "suspicious" byte patterns. For example, long strings of contiguous ASCII characters are unlikely to occur in regular transaction data; similarly, the probability of finding specific bitstrings, like the Gzip header `0x1f9d9070`, is extremely low. Finding such a bitstring is a trigger for further investigation. We employed several types of these searches, which we discuss below.

Frequency analysis. The GNU `strings` utility[7] takes a data source as input, and yields as output all of the ASCII plaintext characters found in that source. It provides a flag for filtering out strings of contiguous ASCII characters under a given length. It is possible to run `strings` directly on Bitcoin Core's .dat files, but care must be taken when tuning the filter. Obviously, too low a threshold will yield a huge number of false positives. On the other hand, due to the way inputs and outputs are encoded in transaction data, too high a threshold eliminates plaintext that has been split across multiple transactions or transaction scripts.

While this approach is quite simple, some of the data that we encountered — particularly, the conversations and code embedded into the blockchain by Peter Todd (one of the Bitcoin Core developers) — mention that they are specifically intended to be discovered and extracted via this method. For example, Todd's plaintext uploader is a Python script stored in the blockchain (see row 14 of Table 1). It describes itself as a tool that can "publish text in the blockchain, suitably padded for easy recovery with `strings`".[8] The tool appears to have been used to upload its own source code to the blockchain. Peter Todd's tool takes a text file as input, and uses the P2SH ignored `OP_PUSHDATA` technique (see Section 3.4) to embed the contents of the file into the

[7] `man7.org/linux/man-pages/man1/strings.1.html`

[8] `github.com/petertodd/python-bitcoinlib/blob/master/examples/publish-text.py#L48`

Type	Technique	Field size	Hosted in	UTXO Bloating	1st item	# items	Tot. size	Avg. size
Single	Value	8	Tx output	No	N/A	N/A	N/A	N/A
	Input Sequence	4	Tx input	No	2011/02/25	1,305,372	5,221,488	4
	P2PK-P2PKH	20,33,65	Script	Yes	2013/03/16	66,762	1,335,240	20
	P2SH	520	Script	Yes	2013/04/10	1,578	31,560	20
	OP_RETURN	80	Script	No	2014/03/12	2,903,186	76,700,965	26
	Vanity Address	20	Script	No	N/A	N/A	N/A	N/A
	Coinbase	2–100	Tx input	No	2009/01/03	305,763	18,442,641	60.3
	Total	—	—	—	**2009/01/03**	**4,582,661**	**101,731,894**	**22**
Multi	Multi-signature	Variable	Script	Transient	2013/04/06	15,067	2,926,590	194
	Multi-in/out	Variable	Variable	Variable	2013/03/16	529	4,437,616	8,389
Chains	Tx chains	Variable	Variable	Variable	2013/04/06	60	3,470,870	57,848

Table 2: Statistics about embedding techniques (sizes are in bytes).

input scripts of a single transaction. The reason this encoding lends well to `strings`-based extraction is that it allows large amounts of arbitrary data to be stored with minimal interruption by non-ASCII bytes. Input scripts are stored contiguously in transaction data, meaning that the only necessary interruptions will be the minimal set of Bitcoin script opcodes required to ensure that the transaction is considered valid by the network.

Compared to the other methods, `strings`-based extraction offers the lowest barrier-to-entry. Thus, users encoding large quantities of plaintext data that are intended to be easily discoverable should make note of encoding methods that lend well to this technique.

File signature. Many file formats require the inclusion of specific bytestrings that are common to all files of a given format. For example, many JPEG images begin with the bytestring `0xffd8ffe000104a464946000101`. Similarly, ASCII-armored PGP messages begin with `-----BEGIN PGP MESSAGE-----`. These bytestrings often occur in the header or footer of the file, although there are formats that place them elsewhere. The probability of finding such bytestrings in Bitcoin blocks is exceedingly low, and as such, they provide a useful indicator of embedded data.

We used several tools to detect file signatures present in Bitcoin transactions:

`binwalk` is an extensible tool for discovering valid files embedded into other data [41]. It provides a language for defining file signatures, as well as a large database of pre-defined signatures for common file formats. It also has the ability to carve detected files out of the surrounding binary data. One can produce a number of valid results simply by running `binwalk` on the Bitcoin Core .dat files. However, since the tool is unaware of the Bitcoin block format, it is only suitable for recovering files embedded into a single transaction script.

`binary-grep` searches a collection of input files for a single bytestring specified by the user [37]. It outputs the byte offsets of any matches, and has a simple carving function.

`local-blockchain-parser` provides a grep command that, unlike `binwalk` and `binary-grep`, is aware of the Bitcoin block format [38], skipping the parts of the transaction that cannot embed metadata. For each match, it outputs the block and transaction hashes, script type (input/output), and byte offset.

One of the most successful workflows we discovered for recovering binary files based on file signatures was the following. (i) We ran `binwalk` and/or `binary-grep` on a .dat file, making note of any results that appeared to be true positives. (ii) If there were promising results, we then ran the `binary-grep` subcommand of `local-blockchain-parser` on that .dat file, obtaining the transaction hashes where those results were found. (iii) For each resulting transaction, we manually inspected the transaction graph around it. If it appeared to be an isolated transaction, we ran the `tx-info` subcommand of `local-blockchain-parser`. If it appeared to be a part of a chain, we ran the `tx-chain` subcommand instead. (iv) We inspected the binary output from the previous step, performing manual carving where necessary, and we attempted to ascertain the validity of the results by opening them with applications appropriate to their file type.

Protocol Identifier. Many protocols mark their metadata by writing a specific string in the first few bytes of each chunk, but the exact number of bytes may vary from protocol to protocol. In Section 5 we take advantage of this for associating metadata to protocols. Furthermore, since protocols give a detailed description of the format of the elements produced, in Section 4.2 we distinguish different types of metadata and classify them. Hence, in order to associate metadata to protocols we: (i) search the web for known associations

between identifiers and protocols; (ii) we accordingly classify strings beginning with one of the identifiers obtained. In more details, in the first step we query Google to obtain public identifier/protocol bindings. For instance, since several protocols use the OP_RETURN technique, we execute the query *"Bitcoin* OP_RETURN*"*, that returns ∼26,500 results, and we manually inspect the first few pages of them. Note that a protocol can be associated with more than one identifier (e.g., Stampery, Blockstore), or even do not have any identifier. In this way we obtain 45 protocols associated to 39 identifiers; further, we find several protocols that do not use any identifier (e.g., Diploma, Chainpoint). We also distinguish the main types of metadata produced by protocols (e.g. Text, Hash and Record). The second step is performed by our tool: it associates chunks of metadata to a protocol. The full list of protocols discovered is shown in Table 5; identifiers are listed in Table 4.

Transaction chains. Although all spent transaction outputs in the Bitcoin blockchain naturally form a chain structure, identifying chains containing embedded metadata is not entirely straightforward. A transaction may have certain "giveaway" characteristics that suggest the presence of a chain containing data, such as:

1. One or more unspendable outputs (i.e., OP_RETURN outputs), plus a single spent output. The unspendable output(s) would contain data, while the spent output would be used to continue the chain.
2. One or more unspent outputs (possibly used for a P2PKH embedding), plus a single spent output. The unspent output(s) would contain data, while the spent output would be used to continue the chain.
3. The unspent outputs, if any, contain a tiny amount of Satoshis (such outputs are also known as *dust*). Except for the Bit-Comm protocol, which uses output values to order the data in the output scripts, the funds included into outputs that can never be spent are effectively "burnt", and add no information to the embedded data. This disincentivizes the embedder from including any more value than is strictly necessary to create a valid transaction.
4. The spent output contains a relatively large amount of bitcoins, used to fund further dust outputs in subsequent links in the chain.
5. Preceding or subsequent transactions share a similar structure with the transaction in question. Many of the transaction chains we found appeared to have been constructed with the help of software (e.g. the Python source we extracted). The software we found tends to create strings of transactions sharing a similar format. While it is altogether possible to embed data into chains of dissimilar transactions, they would be difficult to find and complex to decode.

These are helpful clues, but not definitive criteria. In fact, there are many other types of transactions which possess the characteristics described above. For example, payouts from mining pools and Bitcoin casinos often send small amounts of bitcoins to many users at once. These payout transactions are often constructed algorithmically (according to some set of "threshold" rules intended to minimize the impact of the fee on the payout), meaning that preceding and following transactions share a similar structure.

Therefore, it is generally necessary to have some understanding of the embedded data in order to determine whether a given chain is of interest. If a transaction contains a file signature for a file type that is unlikely to fit into the data provided by that transaction, it warrants further investigation.

Extraction of data from transaction chains is relatively easy when using the `local-blockchain-parser` utility. This utility has a `tx-chain` subcommand that takes a single transaction hash and crawls backwards and forwards through the transaction graph, collecting data from the transaction scripts. This data is filtered and permuted to account for the various ways in which transaction chains are constructed. Finally, the data from each transaction are concatenated in the order that they appear in the chain. This process yields a collection of binary files corresponding to the different ways in which data can be embedded into a chain.

4.2 Types of metadata

We associate the successfully reconstructed data items to one of the following types:

Text Users have embedded a significant amount of text, since the very first message by Satoshi Nakamoto. This includes several birthday wishes, love statements, prayers, greetings, developer conversations, and magnet links. Besides user messages, miners usually embed in coinbase transactions messages for Bitcoin-related purposes, to identify their blocks, vote on proposals, announce what features they support. We have also identified two pdf documents: *"Bitcoin: a peer-to-peer electronic cash system"* [48], and *"The first collision for full SHA-1"* [51].

Hash Many users notarize the ownership of documents by embedding their hash on the blockchain (embedding the whole document would be too expensive, because of the required transaction fees). Some protocols notarize several documents with a single piece of metadata; this could be the hash of the sequence

of document hashes, or the root of the Merkle tree of the document hashes.

Financial record A common application of the Bitcoin blockchain is to record the ownership and exchange of digital or physical assets. These assets are represented as tokens, and users are identified by their Bitcoin addresses.

Copyright Copyright records are produced by protocols which act as marketplaces where artists publish and sell their files to other users.

Script Developers have embedded in the blockchain several scripts. We have found some Python scripts (e.g., the *Satoshi uploader*, *Satoshi downloader*, and *Cryptograffiti uploader*), Bash scripts (e.g., *Password script*, and *OpenSSL encoder*), and also some games (e.g., *LinPyro*, *Bong ball* and *Lucifer*).

Image The blockchain contains some small images, usually spread across chains of transactions. We have found various file formats (PNG, JPEG, and GIF).

Archive This type includes compressed archives, like e.g. the WikiLeaks Cablegate gzipped archive.

4.3 Statistics on types of metadata

Table 3 shows some statistics about the type of metadata we reconstructed. The second column indicates the day in which the first piece of metadata of the corresponding type appeared in the blockchain. Next, we show the total number of elements found, followed by their total size in bytes, and their average size.

Note that the total size of reconstructed metadata is less than the 101,731,894 reported in Table 2: this is because Table 2 also includes bitstrings that we did not manage to decode. For instance, the bytes embedded with `OP_RETURN` are always considered in Table 2, but they appear in Table 3 only if we are also able to recognize their type (e.g., because their prefix reveals which protocol has produced them, among those in Table 4). From the rightmost column we see that the average size of scripts, images and archives exceeds the maximum size of Bitcoin scripts; hence, these metadata are embedded through "Multi-in/out" or "Tx chains" techniques. Finally, note that although the first financial record appeared only in May 2014, this type of metadata now constitutes ~70% of the reconstructed elements, and it uses the majority of the space.

5 Analysis of Bitcoin-based protocols

In this section we focus on protocols which embed metadata on the Bitcoin blockchain. We first propose a rough taxonomy of protocols, which categorize them according to the application domain. Then, considering the collection of protocols reported in a previous paper [34], we perform several analyses on their usage of metadata.

5.1 Types of protocols

Our taxonomy classifies protocols in five categories:

Financial includes protocols that manage assets, e.g. for certifying their ownership, endorsing their value, and keeping track of trades. Metadata in these transactions are used to specify the value of the asset, the amount of the asset transferred, the new owner, *etc.*

Notary includes protocols that certify the ownership and timestamp of documents. These protocols allow users to publish the hash of a document in a transaction, thus proving its existence and integrity. Since the transaction is signed with a private key, users can also certify the ownership of the document.

DRM includes protocols for declaring access rights and copyrights on digital art documents, like e.g. images or audio files.

Message groups protocols which record text messages.

Subchain gathers protocols which construct transaction chains to record execution traces of third-party smart contracts.

We now briefly comment this taxonomy. Although **Notary** and **DRM** protocols have the same overall goal — certifying the ownership of documents — they have some relevant differences. First, **Notary** protocols do not usually require the original document (yet, they ask that the document hash is provided by the owner); further, their goal can be fulfilled also when their front-end is no longer online. Conversely, **DRM** protocols usually need to gather user documents, and have complex front-ends to enable further interactions with users (e.g., they often play the role of broker between media producers and consumers). The ordering of metadata embedded by **Notary**, **DRM** and **Message** protocols is immaterial; instead, different orderings in **Financial** and **Subchain** protocols usually imply different system states. Indeed, transactions used by **Financial** protocols are analogous to Bitcoin transactions, except that they transfer tokens instead of bitcoins; depending on the current balance of tokens, appending a transaction may result in a state update, or even leave the state unchanged (e.g., if an attacker attempts to sell assets that she does not currently own). **Subchain** protocols share the same mechanism, but they generalize token exchange to more complex computations, like those arising from the execution of smart contracts (e.g., in the RSK platform).

Type	1st item	# items	Tot. size	Avg. size
Text	2009/01/03	309,894	18,811,329	61
Hash	2013/12/18	200,832	7,617,392	38
Financial record	2014/05/03	1,430,071	37,699,809	26,36
Copyright	2014/12/19	116,406	3,503,170	30
Script	2013/04/06	10	138,149	13,815
Image	2013/03/17	108	1,523,529	14,107
Archive	2013/04/06	12	2,838,760	236,563
TOTAL	**2009/01/03**	**2,054,575**	**72,132,138**	**35**

Table 3: Statistics on types of metadata (sizes are in bytes).

5.2 Statistics on Bitcoin-based protocols

Table 5 shows some detailed statistics about protocols. The first and second columns indicate, respectively, the protocol type and name. We use an additional type, called **Empty**, to gather the transactions which use `OP_RETURN` without embedding any metadata. The third and fourth columns show the type of metadata and the embedding technique. The fifth column shows when the protocol generated the first chunk of metadata; since transactions do not carry a timestamp, to this purpose we use the timestamp of the enclosing block. The next two columns count the total number of elements produced by a protocol, and the total size (in bytes) of the embedded metadata (net of script instructions and other transaction fields). The rightmost column shows the average size of the metadata.

We were able to associate to protocols ∼ 53.7MB of metadata, which is quite less than the total amount extracted (∼ 99MB). This difference has various motivations. First, users often embed metadata not related to any protocol; for instance, this is the case for several images and text messages. Second, several protocols make it impossible, for an external observer, to recognize their chunks of metadata (unlike the protocols in Table 4, which append an identifier to the metadata): indeed, we have discovered 19 protocols that embed metadata without any identifier. Finally, our list of protocols may be incomplete, so if some other protocols embed metadata with `OP_RETURN`, we count their items but we can not classify them. We note a relevant component of **Empty** transactions (∼10% of the total `OP_RETURN` transactions), which use `OP_RETURN` without any data attached, so they are not associated to any protocol. We evaluate that ∼ 96% of these transactions are related to the peaks, discussed later on in Section 6.4. The fifth column of Table 5 suggests that, originally, the protocols were of **Financial** and **Notary** type, while the other use cases were introduced subsequently (indeed, the others types were not inhabited before the end of 2014).

Type	Protocol	Identifiers
Financial	Colu	CC
	CoinSpark	SPK
	OpenAssets	OA
	Omni	omni
	Openchain	OC
	Helperbit	HB
	Counterparty	CNTRPRTY
Notary	Factom	Factom!!, FACTOM00, Fa, FA
	Stampery	S1, S2, S3, S4, S5, S6
	Proof of Existence	DOCPROOF
	Blocksign	BS
	CryptoCopyright	CryptoTests-, CryptoProof-
	Stampd	STAMPD##
	BitProof	BITPROOF
	ProveBit	ProveBit
	Remembr	RMBd, RMBe
	OriginalMy	ORIGMY
	LaPreuve	LaPreuve
	Nicosia	UNicDC
	SmartBit	SB.D
	Notary	Notary
DRM	Monegraph	MG
	Blockai	0x1f00
	Ascribe	ASCRIBE
Message	Eternity Wall	EW
	BitAlias	BALI
Subchain	Blockstore	id, 0x5888, 0x5808

Table 4: Protocol identifiers. Counterparty metadata must be first deobfuscated with ARC4 encryption, using the transaction identifier of the first unspent transaction output as the encryption key.

From Table 5 we see that the large majority of protocols use the `OP_RETURN` technique. Focussing on the metadata embedded with this technique, Figure 2 displays how metadata are distributed into the protocol types, and Figure 3 shows the temporal evolution of their usage, in terms of the number of metadata items published per week. Comparing Table 5 with Figure 2 we see that although most protocols are **Notary**, their transactions are a fraction of those produced by **Financial** protocols.

Type	Protocol	Metadata	Technique	1st item	# items	Tot. size	Avg. size
Financial	Colu	Financial record	OP_RETURN	2015/07/09	244,411	4,425,702	18
	CoinSpark	Financial record	OP_RETURN	2014/07/02	28,120	960,664	34
	OpenAssets	Financial record	OP_RETURN	2014/05/03	207,132	3,255,499	16
	Omni	Financial record	OP_RETURN	2015/08/10	311,605	6,249,883	20
	Openchain	Hash	OP_RETURN	2015/10/21	2,758	115,283	42
	Helperbit	Financial record	OP_RETURN	2015/09/18	33	1,251	38
			OP_RETURN	2014/06/16	636,012	22,806,810	36
	Counterparty	Financial record	P2PKH	N/A	N/A	N/A	N/A
			Multi-signature	N/A	N/A	N/A	N/A
	Total	—	—	**2014/06/16**	**1,430,071**	**37,815,092**	**26**
Notary	Factom	Merkle root	OP_RETURN	2014/04/11	105,188	4,207,262	40
	Stampery	Merkle root, Hash	OP_RETURN	2015/03/09	74,887	2,648,102	35
	Proof of Existence	Hash	OP_RETURN	2014/04/21	5,464	218,513	40
	Blocksign	Hash	OP_RETURN	2014/08/04	1,477	55,676	38
	CryptoCopyright	Hash	OP_RETURN	2014/08/02	46	1,840	40
	Stampd	Hash	OP_RETURN	2015/01/03	562	22,427	40
	BitProof	Hash	OP_RETURN	2015/02/25	770	30,800	40
	ProveBit	Hash	OP_RETURN	2015/04/05	57	2,280	40
	Remembr	Hash	OP_RETURN	2015/08/25	28	1,128	40
	OriginalMy	Hash	OP_RETURN	2015/07/12	126	4,788	38
	LaPreuve	Hash	OP_RETURN	2014/12/07	68	2,663	39
	Nicosia	Hash of hashes	OP_RETURN	2014/09/12	24	840	35
	SmartBit	Merkle root	OP_RETURN	2015/11/24	8,472	304,992	36
	Notary	Hash	OP_RETURN	2017/04/11	21	798	38
	Originstamp	Hash of hashes	(Commit metadata)	2013/12/18	905	0	0
	Btproof	Hash	(Commit metadata)	N/A	N/A	N/A	N/A
	BitcoinTimestamp	Hash	Value, Multi-in/out	N/A	N/A	N/A	N/A
	Blocknotary	Merkle root	OP_RETURN	N/A	N/A	N/A	N/A
	Tangible	Hash	OP_RETURN	N/A	N/A	N/A	N/A
	Chainpoint	Merkle root	OP_RETURN	N/A	N/A	N/A	N/A
	Diploma	Hash	OP_RETURN	N/A	N/A	N/A	N/A
	Apertus	Hash	P2PKH	N/A	N/A	N/A	N/A
	Chronobit	Hash	N/A	N/A	N/A	N/A	N/A
	Seclytics	Hash	OP_RETURN	N/A	N/A	N/A	N/A
	Total	—	—	**2013/12/18**	**198,095**	**7,502,109**	**38**
DRM	Monegraph	Copyright	OP_RETURN	2015/06/28	67,286	2,464,282	37
	Blockai	Copyright	OP_RETURN	2015/01/09	670	38,327	57
	Ascribe	Copyright	OP_RETURN	2014/12/19	48,450	1,000,561	21
	Verisart	Merkle root	N/A	N/A	N/A	N/A	N/A
	Total	—	—	**2014/12/19**	**116,406**	**3,503,170**	**30**
Message	Eternity Wall	Text	OP_RETURN	2015/06/24	4,129	177,916	43
	Cryptograffiti	Text	P2PKH, Multi-in/out	N/A	N/A	N/A	N/A
	BIT-COMM	Text	P2PKH, Multi-in/out	N/A	N/A	N/A	N/A
	Stone	Text, File	P2PKH, Multi-in/out	N/A	N/A	N/A	N/A
	Key.run	Magnet link	OP_RETURN	N/A	N/A	N/A	N/A
	BitAlias	Secret, Hash	OP_RETURN	—	0	0	0
	Total	—	—	**2015/06/24**	**4,129**	**177,916**	**43**
Subchain	Keybase	Merkle root	OP_RETURN	N/A	N/A	N/A	N/A
	Uniquebits	PGP signed hash	P2PKH, P2SH	N/A	N/A	N/A	N/A
	Blockstore	Key-Value	OP_RETURN	2014/12/10	209,422	6,068,584	29
	Catena [53]	Text	OP_RETURN, Tx chains	N/A	N/A	N/A	N/A
	Total	—	—	**2014/12/10**	**209,422**	**6,068,584**	**29**
Empty	**Total**	—	OP_RETURN	**2014/03/20**	**296,396**	**0**	**0**
TOTAL	—	—	—	**2009/01/03**	**2,254,519**	**55,066,871**	**24**

Table 5: Statistics on Bitcoin-based protocols (sizes are in bytes).

6 Discussion

In this section we discuss the impact of metadata on the Bitcoin blockchain. We start by describing the historical evolution of metadata, highlighting how the adoption of the embedding techniques has varied over the years. We then evaluate the memory and storage consumption due to metadata, and we discuss the phenomenon of transaction peaks.

6.1 Historical perspective

The first piece of metadata was embedded in the genesis block by Satoshi Nakamoto, through the Coinbase technique; then, since October 2011, this technique has been used regularly by miners. In the first 3 years of Bitcoin, the most used technique for embedding data was P2PKH. Later, many protocols (e.g., Counterparty) migrated from the P2PKH to the OP_RETURN technique, and we rarely find protocols still using P2PKH. The P2PKH is now used for embedding large files with the

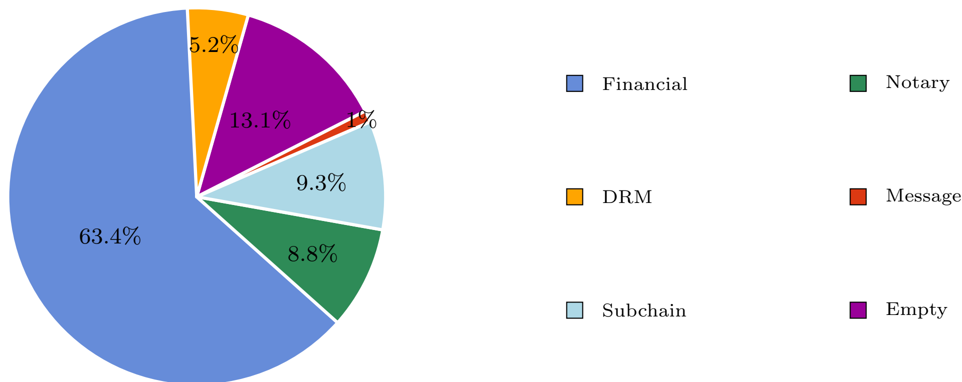


Fig. 2: Metadata transactions by protocol type.

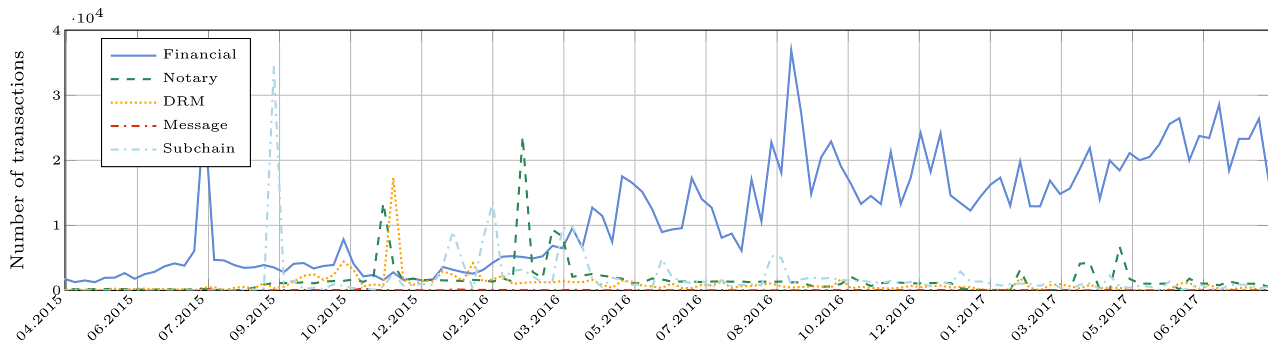


Fig. 3: Temporal evolution of metadata (transactions before April 2015 are negligible).

support of the Multi-in/out, Multi-signature, and Tx chains techniques. Despite the similarity with P2PKH, the P2PK technique is less used, since P2PK scripts are considered obsolete [10]. The Input Sequence and the Value techniques are not widely adopted as well, probably because of the limited space they offer respect to other techniques. Also the P2SH technique is not widely used, although there are some proposals for adopting it for Counterparty[9].

Although OP_RETURN has been part of the scripting language since the first releases of Bitcoin, originally it was considered non-standard, so transactions containing this opcode were not reliably mined. OP_RETURN became standard with Bitcoin Core 0.9.0 [8], but still the release notes state that: *"This change is not an endorsement of storing data in the blockchain. The OP_RETURN change creates a provably-prunable output, to avoid data storage schemes [...] that were storing arbitrary data such as images as forever-unspendable TX outputs, bloating bitcoin's UTXO database"*. The limit for storing data with OP_RETURN was originally planned to be 80 bytes, but the first official client supporting the opcode, i.e. the release 0.9.0, allowed only 40 bytes. This animated a long debate [4, 5, 13, 15]. From the release 0.10.0 [6] nodes could choose whether to accept or not OP_RETURN transactions, and set a maximum for their size. The maximum size was then set to 80 bytes by the release 0.12.0 [7]. From Table 5 we see that the majority of the applications built on top of Bitcoin embed metadata through the OP_RETURN technique; this is coherent with the data in Table 2, from which we see that ∼63% of the metadata in the blockchain have been embedded with the OP_RETURN technique (which is the most adopted one since March 2014). In the last period of our experiments, ∼40,000 new OP_RETURN transactions are published each week. Overall, OP_RETURN transactions amount to $\sim 1,18\%$ of the total number of transactions ($\sim 1,37\%$ when considering the portion of the blockchain from 2014/03/12, when the first OP_RETURN transaction appeared)[10].

[9] See counterpartytalk.org/t/cip-proposal-p2sh-data-encoding/2169.

[10] Despite the 5 years of delay, this percentage is quite close to that for the whole blockchain: this is because the number of daily transactions has largely increased since July 2014.

6.2 UTXO bloating

As remarked in Section 3, the embedding techniques P2PK, P2PKH, and P2SH (which are often used to embed media files), produce unspendable outputs. In this way they contribute to the "UTXO bloating" effect, that deteriorates the performance of Bitcoin nodes. In the UTXO set we have counted ∼68K unspendable outputs which are used to embed chunks of metadata. Of all the transaction which embed metadata, only ∼1.49% contribute to the UTXO bloating effect.

The other embedding techniques, among which the `OP_RETURN`, do not bloat the UTXO (even though they still affect the total size of the blockchain). This, together with the possibility of embedding up to 80 bytes of metadata, are perhaps the reasons of the popularity of the `OP_RETURN` technique. Indeed, we see from Table 5 that `OP_RETURN` is used by the large majority of Bitcoin-based protocols. Note that the other techniques which avoid the UTXO bloating effect are not suitable to be used by protocols, either because they have a low bandwidth (Coinbase), or because they do not allow to embed enough bytes (from Table 5 we see that, on average, protocols require to embed 24 bytes of metadata).

6.3 Space consumption

A debated topic in the Bitcoin community is whether it is acceptable or not to save arbitrary data in the blockchain. From Table 5 we can see that the net size of metadata is ∼ 99MB. In same period of observation, the size of the whole blockchain is ∼ 125GB, so the size of metadata amounts to ∼ 0.077% of the total size of transactions.

For the most widespread embedding method, the `OP_RETURN`, Figure 4a shows the average length of the metadata of each week. Generally, the average length of metadata is less than 40 bytes, despite the extension to 80 bytes introduced on 2015/07/12. Peaks down on the same period are related to the **Empty** transactions, discussed later on in Section 6.4. Figure 4b represents the number of `OP_RETURN` transactions with a given data length: also this chart confirms a small number of transactions that use more than the half of the available space. Note that the discussed peak appears also in this chart, in correspondence of the 0 value. From the last column of Table 5 we see that even the protocol which embeds the largest number of bytes (Blockai, with 57 bytes on average), requires much less than the 80 bytes available with `OP_RETURN`. Several **Notary** protocols take 40 bytes on average: 16 bytes for their identifiers, and the remaining bytes for the hash they save. Generally, **Notary** protocols carry longer metadata than the other protocols.

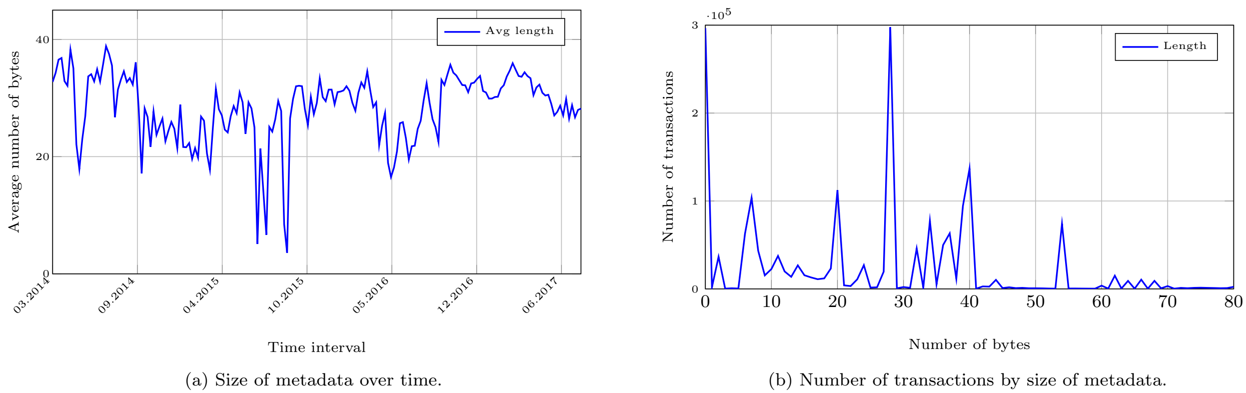
We now estimate the overall size of `OP_RETURN` transactions (including both the metadata and the other parts of the transaction). The size of an **Empty** transaction with one input and one output is 156 bytes. From Table 2 we see that `OP_RETURN` transaction carry 26 bytes of metadata, on average. We then approximate the average size of an `OP_RETURN` transaction as 182 bytes. Multiplying by the number of `OP_RETURN` transaction, we obtain an approximation of their space consumption as ∼ 503MB.

6.4 Transaction peaks

Figure 5 represents peaks of `OP_RETURN` transactions from 2014/03 (date of the first `OP_RETURN` transaction) to 2017/08. For each week, it shows (i) the number of **Empty** transactions, (ii) the number of `OP_RETURN` transactions which are not produced to any protocol (among those in our collection), and (iii) the total number of `OP_RETURN` transactions. In the graph we note several peaks, that we explain as follows:

1. ∼100K transactions from 2015/07/08 to 2015/08/05. This peak is mainly composed of two different peaks of **Empty** transactions: the July peak (∼37K transactions from 2015/07/08 to 2015/07/10) and the August peak (∼29K transactions from 2015/08/01 to 2015/08/03). Both peaks occurred coincidentally with stress tests and spam campaigns [32][11].
2. ∼300K transactions from 2015/09/09 to 2015/09/23. This second peak is the highest and longest-lasting one. As before, it is mainly caused by **Empty** transactions (∼223K), although here we also observe a component of **Unclassified** and Blockstore transactions (∼35K each). The work [32] detects a spike also in this period, precisely around 2015/09/13, where an anonymous group performed a stress-test on the network with a *money drop*. This involves a public release of private keys, with the aim to cause a big race and a consequent large number of *double-spend* transactions. More specifically, people used the private keys to transfer to themselves the bitcoins redeemable with these keys; since many people tried to perform these transfers simultaneously, the network was flooded with many transactions trying to double-spend the same outputs. The confirmed

[11] We conjecture that **Empty** transactions are caused by these events. To verify this conjecture we would need to compare the transaction identifiers of our **Empty** transactions with the identifiers of [32], which are not publicly available.



(a) Size of metadata over time.

(b) Number of transactions by size of metadata.

Fig. 4: Usage and size of OP_RETURN transactions.

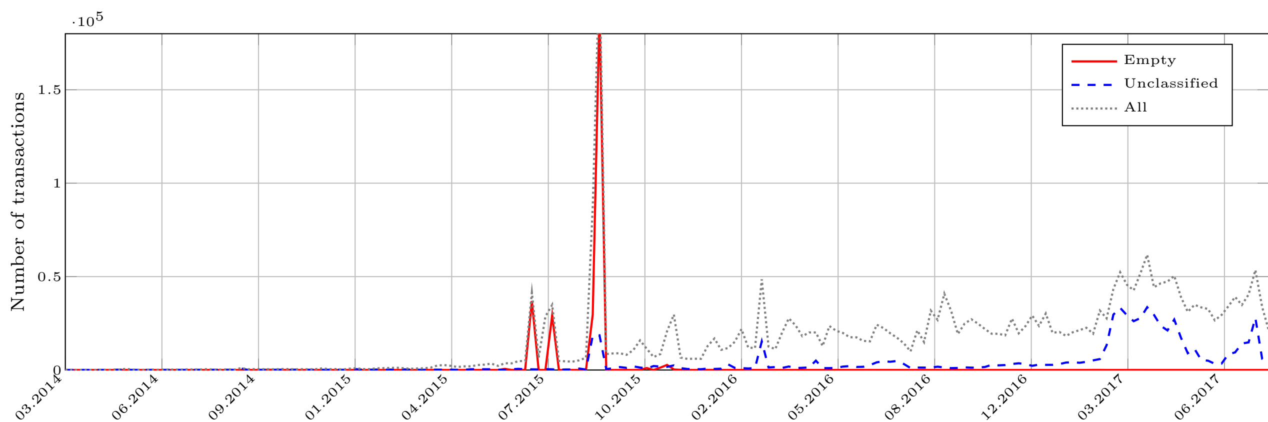


Fig. 5: Transactions (and transactions peaks) over time.

transactions caused a peak, which happened simultaneously to the peak of OP_RETURN we measured.

3. ∼50K transactions from 2016/03/02 to 2016/03/09. This last peak is given by the sum of two different peaks: **Unclassified** (∼18K) and Stampery (∼23K) transactions. The part of the peak caused by Stampery can be explained as follows. Being a notarization protocol, Stampery receives document hashes off-chain, and subsequently it embeds these hashes in transactions. Since Stampery has only a few transactions before 2016/03/02 (probably, used for testing), we conjecture that the peak coincides with its bootstrap, when the protocol publishes on the blockchain all the transactions related to the hashes accumulated off-chain. The other part of the peak could be due to the bootstrap of other protocols.

Besides the peaks of OP_RETURN transactions, we can also observe other peaks: for instance, for a duration of 100 blocks starting from 2015/05/22, Bitcoin was targeted by a stress test [2], during which the network was flooded with a large number of transactions. However, the usage of OP_RETURN transactions in this period does not seem to deviate from their normal usage.

7 Related works

There is a growing literature on the analysis of the Bitcoin blockchain [32, 45, 47, 49, 50], and also some online services which perform statistics on Bitcoin metadata [3, 17, 20, 24]. Below, we group the related works into three categories.

The first category includes online services related to Bitcoin metadata. The website opreturn.org shows some statistics about OP_RETURN transactions, organised by protocol, and statistics about their usage in a certain time frame. The website smartbit.com recognises some OP_RETURN protocols and shows statistics on them. Finally, the website kaiko.com sells data about OP_RETURN transactions.

The second category contains the works on embedding techniques. At the best of our knowledge, besides our work, this category includes only [26, 46], which have been developed concurrently and independently from ours. Despite the common goals, the works [26,46] differ from ours in several aspects: (i) the “Tx chains” methods and the techniques for committing metadata are described only in our work; (ii) only our work and

[46] extract and quantify the embedded metadata; (iii) the P2SH techniques are detailed in [26]. Further differences between our work and [46] are discussed below.

The third category includes the works which analyse the types of metadata, as those in Section 4. Also in this case, the work [46] is the closest to ours: the main difference between the two works is that, while [46] is focussed on discussing the benefits and risks related to metadata (e.g. privacy violations, illegal contents), we develop a protocol-wise analysis, measuring how much (and when) metadata is embedded by each protocol, and studying which use cases they support. Further, we recognize a few types of metadata (hash, financial records, and copyright) which are not dealt with by [46].

8 Conclusions

Although Bitcoin does not explicitly support for embedding metadata into transactions, over the years users have devised various techniques to reach this goal. After illustrating and comparing these techniques, we have extracted all the metadata embedded up to 2017/08/10 (first 480,000 blocks), measuring the data stored by each technique. By processing the bytes extracted from transaction metadata, we have often managed to reconstruct the original content. Overall, we have reconstructed ∼ 69MB of documents, out of the ∼ 99MB totally embedded. We have classified these documents, finding that the majority of them are records produced by financial protocols. We have reconstructed also 120 files of various kinds (among which, 108 images), for a total size of ∼ 4MB.

We have discovered 45 protocols which embed metadata into the blockchain for developing various applications. We have identified which types of metadata they produce, and which embedding techniques they use. Usually, each protocol produces one type of metadata (depending on the protocol type), using one embedding technique (most often, `OP_RETURN`). Overall, ∼ 53.7MB of metadata are produced by the protocols in our collection. The majority of protocols are for document notarization, but ∼70% of elements are produced by financial protocols.

Finally, we have discussed the impact of embedding metadata in the blockchain, considering various aspects, like e.g. the space consumption, the UTXO bloating effect, and the transaction peaks.

Although the official Bitcoin documentation discourages the use of the blockchain to store arbitrary data, the trend seems to be a growth in the number of applications that embed their metadata in Bitcoin transactions. We conjecture that the perceived sense of security and persistence of the Bitcoin blockchain is the main motivation to avoid using cheaper and more efficient storage. If this trend will be confirmed, the specific needs of these applications could affect the future evolution of the Bitcoin protocol.

Acknowledgements We thank Nicola Atzei for the insightful discussion on a preliminary version of this paper. This work is partially supported by Aut. Reg. of Sardinia projects *Sardcoin* and *Smart collaborative engineering*, and by *COST Action IC1406 cHiPSET*.

References

1. Bicoin scalability, https://en.bitcoin.it/wiki/Scalability_FAQ. Last accessed 2018/01/01
2. Bitcoin network survives surprise stress test, http://www.coindesk.com/bitcoin-network-survives-stress-test/. Last accessed 2018/01/01
3. Bitcoin OP_RETURN wiki page, https://en.bitcoin.it/wiki/OP_RETURN. Last accessed 2018/01/01
4. Bitcoin pull request 5075, https://github.com/bitcoin/bitcoin/pull/5075. Last accessed 2018/01/01
5. Bitcoin pull request 5286, https://github.com/bitcoin/bitcoin/pull/5286. Last accessed 2018/01/01
6. Bitcoin release 0.10.0, https://bitcoin.org/en/release/v0.10.0. Last accessed 2018/01/01
7. Bitcoin release 0.12.0, https://bitcoin.org/en/release/v0.12.0. Last accessed 2018/01/01
8. Bitcoin release 0.9.0, https://bitcoin.org/en/release/v0.9.0. Last accessed 2018/01/01
9. Bitcoin script interpreter, https://github.com/bitcoin/bitcoin/blob/fcf646c9b08e7f846d6c99314f937ace50809d7a/src/script/interpreter.cpp. Last accessed 2018/01/01
10. Bitcoin wiki script, https://en.bitcoin.it/wiki/Script. Last accessed 2018/01/01
11. Bitcoin wiki transaction, https://en.bitcoin.it/wiki/Transaction. Last accessed 2018/01/01
12. Colu website, https://www.colu.com/. Last accessed 2018/01/01
13. Counterparty open letter and plea to the Bitcoin core development team, http://counterparty.io/news/an-open-letter-and-plea-to-the-bitcoin-core-development-team/. Last accessed 2018/01/01
14. Counterparty website, http://counterparty.io/. Last accessed 2018/01/01
15. Developers battle over bitcoin block chain, http://www.coindesk.com/developers-battle-bitcoin-block-chain/. Last accessed 2018/01/01
16. Factom website, https://www.factom.com/. Last accessed 2018/01/01
17. Kaiko data store, https://www.kaiko.com/. Last accessed 2018/01/01
18. Omni website, http://www.omnilayer.org/. Last accessed 2018/01/01
19. Open assets website, https://github.com/OpenAssets/. Last accessed 2018/01/01
20. opreturn.org, http://opreturn.org/. Last accessed 2018/01/01
21. Proof of existence website, https://proofofexistence.com/. Last accessed 2018/01/01

22. Scalability debate ever end, https://www.cryptocoinsnews.com/will-bitcoin-scalability-debate-ever-end/. Last accessed 2018/01/01
23. Scaling debate in Reddit, http://www.coindesk.com/viabtc-ceo-sparks-bitcoin-scaling-debate-reddit-ama/. Last accessed 2018/01/01
24. Smartbit OP_RETURN statistics, https://www.smartbit.com.au/op-returns. Last accessed 2018/01/01
25. Stampery blockchain timestamping architecture, https://s3.amazonaws.com/stampery-cdn/docs/Stampery-BTA-v6-whitepaper.pdf. Last accessed 2018/01/01
26. Data insertion in Bitcoin's blockchain (2017), http://digitalcommons.augustana.edu/cgi/viewcontent.cgi?article=1000&context=cscfaculty. Last accessed 2018/01/01
27. Andresen, G.: Block v2, height in coinbase, BIP 034, https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki. Last accessed 2018/01/01
28. Antonopoulos, A.M.: Mastering Bitcoin: unlocking digital cryptocurrencies. O'Reilly Media, Inc. (2014)
29. Atzei, N., Bartoletti, M., Cimoli, T., Lande, S., Zunino, R.: SoK: unraveling Bitcoin smart contracts. In: Principles of Security and Trust (POST). LNCS, vol. 10804, pp. 217–242. Springer (2018)
30. Atzei, N., Bartoletti, M., Lande, S., Zunino, R.: A formal model of Bitcoin transactions. In: Financial Cryptography and Data Security (2018)
31. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: CRYPTO. LNCS, vol. 10401, pp. 324–356. Springer (2017)
32. Baqer, K., Huang, D.Y., McCoy, D., Weaver, N.: Stressing out: Bitcoin "stress testing". In: Financial Cryptography Workshops. LNCS, vol. 9604, pp. 3–18. Springer (2016)
33. Bartoletti, M., Bracciali, A., Lande, S., Pompianu, L.: A general framework for blockchain analytics. In: Proc. 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL@Middleware). pp. 7:1–7:6. ACM (2017), https://github.com/bitbart/blockapi
34. Bartoletti, M., Pompianu, L.: An analysis of Bitcoin OP_RETURN metadata. In: Financial Cryptography Workshops. LNCS, vol. 10323, pp. 218–230. Springer (2017)
35. Bartoletti, M., Pompianu, L., Bellomy, B.: Bitcoin metadata (2018), https://doi.org/10.7910/DVN/MOLW81
36. Bartoletti, M., Zunino, R.: BitML: a calculus for Bitcoin smart contracts. In: ACM CCS (2018)
37. Bellomy, B.: Binary grep, https://github.com/spooktheducks/binary-grep
38. Bellomy, B.: Local blockchain parser, https://github.com/spooktheducks/local-blockchain-parser
39. Bistarelli, S., Mercanti, I., Santini, F.: An analysis of non-standard Bitcoin transactions. In: Crypto Valley Conference on Blockchain Technology (2018)
40. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In: IEEE Symp. on Security and Privacy. pp. 104–121 (2015)
41. devttys0: binwalk, https://github.com/devttys0/binwalk
42. Garay, J.A., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT. LNCS, vol. 9057, pp. 281–310. Springer (2015)
43. Gerhardt, I., Hanke, T.: Homomorphic payment addresses and the pay-to-contract protocol. arXiv preprint arXiv:1212.3257 (2012)
44. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE Symp. on Security and Privacy. pp. 839–858 (2016)
45. Lischke, M., Fabian, B.: Analyzing the Bitcoin network: The first four years. Future Internet 8(1), 7 (2016)
46. Matzutt, R., Hiller, J., Henze, M., Ziegeldorf, J.H., Müllmann, D., Hohlfeld, O., Wehrle, K.: A quantitative analysis of the impact of arbitrary blockchain content on Bitcoin. In: Financial Cryptography and Data Security (2018)
47. Möser, M., Böhme, R.: Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees. In: Financial Cryptography and Data Security. LNCS, vol. 8976, pp. 19–33. Springer (2015)
48. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
49. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: Security and privacy in social networks, pp. 197–223. Springer (2013)
50. Ron, D., Shamir, A.: Quantitative analysis of the full Bitcoin transaction graph. In: Financial Cryptography and Data Security. LNCS, vol. 7859, pp. 6–24. Springer (2013)
51. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: CRYPTO. LNCS, vol. 10401, pp. 570–596. Springer (2017)
52. Todd, P.: Delayed TXO commitments, https://petertodd.org/2016/delayed-txo-commitments. Last accessed 2018/01/01
53. Tomescu, A., Devadas, S.: Catena: Efficient non-equivocation via Bitcoin. In: IEEE Symp. on Security and Privacy. pp. 393–409 (2017)