Short Paper: An Analysis of Non-standard Bitcoin Transactions

Stefano Bistarelli*, Ivan Mercanti* and Francesco Santini*
Dipartimento di Matematica e Informatica, University of Perugia, Italy
Email: *[firstname.lastname]@unipg.it

***Abstract*—In Bitcoin, the most common kind of transactions is in the form "Bob pays Alice", and it is based on the *Pay to-Public Key Hash* (*P2PKH*) script. P2PKH transactions are just one among many *standard* classes: a transaction is standard if it passes *BitcoinCore*'s *IsStandard()* and *IsStandardTx()* tests. However, the creation of ad-hoc scripts to lock (and unlock) transactions allows for also generating *non-standard* transactions, which can be nevertheless broadcast and mined as well. In this work, we explore the Bitcoin block-chain with the purpose to analyse and classify standard and non-standard transactions, understanding how much the standard behaviour is respected.**

I. Introduction

The white-paper on Bitcoin appeared in November 2008 [7], written by a computer programmer(s) using the pseudonym "Satoshi Nakamoto". His invention is an open-source, peer-to-peer digital currency. Money transactions do not require a third-party intermediary, with no traditional financial-institution involved in transactions. Therefore, the Bitcoin network is completely decentralised, with all the parts of transactions performed by the users of the system.

In this paper we investigate *standard* and *non-standard* transactions in the block-chain of Bitcoin. Transactions are standard if they pass the controls implemented in the reference Bitcoin client, i.e., *Bitcoin Core*. Our interest is mainly focused on non-standard ones, of which we provide a classification in nine different types. The motivation consists in understanding what and how Bitcoin-protocol flaws have been exploited so far, accidentally or not. Hence, we can evaluate the errors and misuses accepted by some of the miners in the network: only 0.03% of transactions is non-standard (2009 - March 2018).

II. Background

A. Transactions

A Bitcoin *wallet* stores a collection of public/private key-pairs of a user, and not directly bitcoins. A Bitcoin address is an identifier of 26-35 alphanumeric characters, and it strictly derives from the hash of a generated public key (*pubkey* in the following) [2]. A private key is a random 256bit number, and the corresponding pubkey is generated through an *Elliptic Curve Digital Signature Algorithm* (*ECDSA*). A transaction *input* must store the proof it belongs to who wants to reuse the money received in a previous transaction. The *output* of a transaction describes the destination of bitcoins instead. So the ownership of the coins is expressed and verified through links to previous transactions. For example, in order to send 3 bitcoins (BTC) to Bob, Alice needs to refer to other transactions she has previously received, whose amount is 3 BTC at least. To lock the coin, a script called *scriptPubKey* is used, while to prove the ownership of a coin, a script called *scriptSig* is used instead. In the following, we will refer to them as "locking script" and "unlocking script".

B. Scripting Language

The bitcoin transactions language *Script* is a Forth-like [8] stack-based execution language. Script requires minimal processing and it is intentionally not Turing-complete (no loops) to lighten and secure the verification process of transactions. An interpreter executes a script by processing each item from left to right in the script. Script is a stack-based language: data is pushed onto the stack, as well as operations, which can push or pop one or more parameters onto/from the execution stack, operate on them and possibly push their result onto the stack.

For example, the operator OP_ADD pops two items from the stack, add them, and finally push the resulting sum onto the stack [2]. There are also conditional operators as OP_EQUAL: it pops two items from the stack and pushes TRUE (represented by number 1) if operands are equal, or FALSE (represented by 0) if they are not equal. In Bitcoin, transaction scripts usually contain a final conditional operator, so that they can produce the result TRUE, which points to a valid transaction.

Most locking scripts refer to a public key address: they require the proof of ownership of the address in order to spend money. However, this is not mandatory [1]: any combination of locking and unlocking scripts that result in a final TRUE value is valid. For example we can have this locking script: "2 OP_ADD 8 OP_EQUAL", which can be satisfied by the unlocking script: "6"'. The validation software combines the locking and unlocking scripts and produces the following script: "6 2 OP_ADD 8 OP_EQUAL". This script is interpreted left-to-right as: first 8 is pushed onto an empty stack, then 2, and then the operation OP_ADD is performed between the two last operands in the stack, which are also popped from it. The result, i.e. 8, is pushed onto the stack, and then the 8 in the script is pushed as well. Finally, OP_EQUAL is performed, removing the two 8 and pushing TRUE.

III. Statistics of Bitcoin transactions

In this section we describe standard and non-standard transactions[1] in the Bitcoin block-chain and we report statistics on their number. To accomplish such an analysis we take advantage of a PostgreSQL Database[2] in which we have

[1]http://www.quantabytes.com/articles/a-survey-of-bitcoin-transaction-types.
[2]PostGreSQL: https://www.postgresql.org.

978-1-5386-7204-4/18/$31.00 ©2018 IEEE
DOI 10.1109/CVCBT.2018.00016

included the blocks up to number $516,040$, that is until April the 1st 2018. We consider Bitcoin Core 0.9[3] as the reference implementation.

A. Standard Transactions in block-chain

In the first few years of Bitcoin history, the developers introduced some limitations in the scripts that could be processed by the reference client. In fact, transactions can be accepted by the network if their locking and unlocking scripts match a small set of believed-to-be-safe templates. This is the *isStandard()* and *isStandardTx()* test, and transactions passing it are called standard transactions.[4] The main reason behind defining and checking standard transactions is to prevent someone from attacking Bitcoin by broadcasting harmful transactions. Moreover, these checks also keep users from creating transactions that would make adding new transaction features in the future more difficult. There are seven standard types of transactions.

Pay to Public Key Hash (P2PKH): The transaction *Pay to public key hash* is the most used in the network. This is because it is the default transaction in a Bitcoin client. These transactions contain a locking script that encumbers the output with a public key hash: "OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG". A P2PKH output can be unlocked (spent) by a public key and a digital signature created with the corresponding private key: "<SIGNATURE A> <PUBLIC KEY A>".

Pay to Public Key (P2PK): The *Pay to Public Key* scheme is type simpler than P2PKH; it is now most often seen in coinbase transactions. P2PK, as the name suggests, has in its locking script directly the pubkey, instead of its hash: "<PUBLIC KEY A> OP_CHECKSIG". To unlock this transaction, only the corresponding signature of pubkey in the locking script is needed: "<SIGNATURE A>".

Multi-signature: *Multi-signature* scripts set a condition where N public keys are recorded in the script, and at least M of those signatures must be used to unlock a transaction. This is also known as an *M-of-N* scheme, where N is the total number of keys and M is the lower threshold of signatures required for a validation. The general form of a locking script setting an *M-of-N* multi-signature condition is: "M <PUBLIC KEY 1> <PUBLIC KEY 2> ... <PUBLIC KEY N> N OP_CHECKMULTISIG". The locking script can be satisfied with an unlocking script containing: "OP_0 <SIGNATURE 1> <SIGNATURE 2> ... <SIGNATURE M>".[5]

Data output (OP_RETURN): The *Data output* transactions are used to store data not related to bitcoin payments [3]. Their form is "OP_RETURN <DATA>". Since any output with OP_RETURN is provably unspendable, they can be used to save different information on the blockchain, which is in this way used as an immutable distributed ledger by applications, as, for example e-voting [4] ones. Such transactions are unlockable.

Pay to Script Hash (P2SH): A *Pay to Script Hash* transactions contain the hash of a script of a different transaction (called *redeem script*) in their locking script. For example, we can hash a *2-of-5* multi-signature transaction. Instead of "pay to this 5-key multi-signature script", the P2SH equivalent transaction is "pay to a script with this hash". Hence, the script only stores a 20-byte hash instead of five pubkeys. In the following, an example of locking script: "OP_HASH160 <2-OF-5 MULTI-SIGNATURE SCRIPT HASH> OP_EQUAL". To unlock coins, the original script and all the necessary signatures are required. For two signatures: "<SIG1> <SIG2> <2-OF-5 MULTI-SIGNATURE SCRIPT>".

Pay to Witness Public Key Hash (P2WPKH): With the introduction of the Segregated Witness[6] (*SegWit*) in Bitcoin, the default transaction P2PKH can be also obtained in a different way. In fact, in place of the longer script in P2PKH, the script in P2WPKH is shortened to: "OP_0 <PUBLIC KEY A HASH>" A P2WPKH output can be unlocked (spent), as the P2PKH, by a public key and a digital signature created by the corresponding private key: "<SIGNATURE S> <PUBLIC KEY A>". The difference is that these are no longer in the unlocking script, but in the witness field instead.

Pay to Witness Script Hash (P2WSH): As for P2PKH/P2WPKH, the only difference between P2SH and P2WSH is in the location of what was previously in the unlocking script, and the locking script being modified. In fact, the locking script becomes only the hash of the redeem script: "OP_0 <REDEEM SCRIPT HASH>", and as for P2WPKH, what was previously in the locking script is moved to the witness field.

Statistics for standard transactions: Considering the first $516,040$ blocks in the block-chain, there are $307,844,694$ transactions that generate a total of $837,854,035$ outputs, of which $782,343,118$ have been spent (93.37%). These include $837,632,874$ standard transaction outputs ($99,97$% of the total number of outputs) of which $782,123,115$ spent ($93,37$%). Hence, non-standard transaction outputs are only the 0.03%.

In Fig. 1 we show the distribution of standard transactions. As introduced before, the most common class is represented by P2PKH transactions, since they are the default ones in the Bitcoin client. The P2SH scheme is the second mostly frequently used class of transactions, with more than 100 million outputs. Interestingly, the number of P2SH and OP_RETURN transactions has considerably increased if compared to the data from 2014.[7] We have also noticed that the most used *M-of-N* multi-signature class corresponds to *1-3*, with 57% of all the multi-signature transactions. The second most used class is *1-2*, with 42%.

B. Non-standard transactions in block-chain

Transactions are validated through *isStandard()* and *isStandardTx()* functions in the Bitcoin Core reference implementation. In case they do not pass such tests, they are simply discarded. However, some of them which deviate from the standard enforced by BitcoinCore can be mined as well: these transactions can be issued in the block-chain thanks to miners

[3]https://bitcoincore.org.

[4]To see which transactions are valid, it is possible to check the reference source code of *BitcoinCore*: https://github.com/bitcoin/bitcoin.

[5]The prefix *op_0* is required because of a bug in the original implementation of CHECKMULTISIG: due to this bug, one more argument on the stack is required. CHECKMULTISIG simply considers it as a placeholder.

[6]https://bitcoincore.org/en/2016/01/26/segwit-benefits/.

[7]http://www.quantabytes.com/articles/a-survey-of-bitcoin-transaction-types.

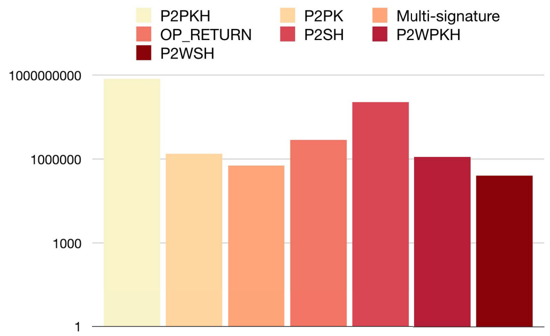

Fig. 1. Distribution of standard transactions.

that relax these checks enforced by such control functions[8]. Non-standard transactions use more complex script forms, represent challenges, or just result from bugs. Their singularity comes from non-standard inputs or outputs. We now revise nine different classes of non-standard transactions (between [], the years in which they have been collected).

OnlyHash [2011-2014]: The *OnlyHash* transactions are the most numerous non-standard class in the block-chain. These transactions contain a hash in the locking script, which is usually the hash of a file for using the block-chain as a ledger to register documents. Therefore, the security and resilience of the block-chain system can be used by applications such as digital-note services, stock exchange certificates, and smart contracts. The blocking script corresponds to "<HASH-OF-SOMETHING>". These transactions were used before the introduction of the OP_RETURN, which can be used for the same purpose. An example is in the output 0 of transaction d65bb24f6289dad27f0f7...989a82dcf5a86fb1c6f8ff2b2c190f[9].

P2Pool Bug [2012]: These transactions are due to a bug[10] in the P2Pool[11] mining tool. Instead of a plain P2PKH locking script, the following script was added: "OP_IFDUP OP_IF OP_2SWAP OP_VERIFY OP_2OVER OP_DEPTH". This script is consequently unlockable. We can see an example of it in output 1 of transaction fa735229f650a8a12bcf2f14...f0aabc52f8687ee148c9f9ab6665[12].

OP_CHECKLOCKTIMEVEIRFY OP_DROP (CLTV) [2012]: In this case the locking script is in the form: "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP". The OP_CHECKLOCKTIMEVEIRFY operator makes the transaction invalid if the element at the top of the stack is greater than the *nLockTime*[13] field of a transaction. In practice, no one can unlock the transaction before a certain date. Therefore, by checking the <DATA> element against the rules above, we can verify the transaction by using an unlocking script that inserts TRUE into the stack. An example of this transaction involves output 1 of transaction 92046da1b339aad23181dbd...e3adf6fc55fca8a705198d9a2324[14] and input 1 in transaction 0157f2eec7bf856d6671485...dc6fa576bec2 00a9fa8039459e78[15].

Pay to Public Key Hash 0 [2011]: It corresponds to a distortion of P2PKH, with the difference that instead of the hash of a pubkey, there is a 0 value. The locking script is in the form "OP_DUP OP_HASH160 0 OP_EQUALVERIFY OP_CHECKSIG". These transactions are unspendable because, as P2PKH ones, in order to verify them a miner needs *i)* the pubkey corresponding to the hash in the locking script, and *ii)* the private key to generate the corresponding signature. However, we know that HASH160 returns a 20 byte long hash: therefore, no key passed to the hash function can return 0. An example is in output 1 of transaction 15ad0894ab42a46eb04108fb8b...d2103f077710733e0516c3a[16].

OP_MIN OP_EQUAL [2012]: These transactions are related to a script as "OP_MIN 3 OP_EQUAL", which simply needs two numbers corresponding to the equation $x \leq y \wedge x = 3$ to be verified. Hence, to unlock it is possible to prepare an unlocking script as "3 4". This means that anyone can easily unlock such a transaction without any private key. An example is in output 1 of transaction aea682d68a3ea5e3583e088...4b083f02ad0aaf2598fe1fa4dfd4[17] and how it has been verified in input 1 of transaction 0f24294a1d23efbb49c1765...2752aba6d765870082fe4f13cae[18].

UnLocked (UL) [2015]: This transaction has an empty locking script, so it can be unlocked by simply having TRUE as unlocking script. Almost all of these transactions carry an amount of 0 BTC, i.e., they are valueless transactions. An example is in output 0 of 87d8ed6aae8ad82b0...55e83e8fa2ae587892f12b06d3a12253c[19]. How it is spent is in input 0 of transaction ce7d73cba662af3dfbd6993...4b3748a88bf9bc70c1cc4d08660[20].

P2PKH NOP [2011, 2014]: This transaction is identical to P2PKH with the only difference that a NOP (an operation that does nothing) is in the locking script: "OP_DUP OP_HASH160 <HASHPUBKEY> OP_EQUALVERIFY OP_CHECKSIG OP_NOP". It can be unlocked by a script identical to that of P2PKH. An example of it is in output 2 of 3077f4b06d7cdb94340...e4331ca3749bdd599d50912f569[21]. How it is verified is in input 2 of 2dc793134d1063a2f2505b...5a707bed6a36f2b71d051aec0[22].

OP_RETURN ERROR [2016-2017]: These transactions are identical to OP_RETURN, with the difference that there is an error in the script code. The code asks to push onto the stack a number of opdcode higher than what is really in the code itself. This is the locking script: "OP_RETURN ERROR" (an error is returned). An example of it is in output 1 of a3247a4e9d249a0dda36c9fc3...eb2c9402ec15ffdd8bdac[23].

[8]http://eligius.st.
[9]https://block-chain.info/it/tx-index/2557393/0.
[10]https://bitcointalk.org/index.php?topic=140097.5;imode..
[11]http://p2pool.in/.
[12]https://block-chain.info/it/tx-index/2915138/1
[13]https://en.bitcoin.it/wiki/NLockTime.
[14]https://block-chain.info/it/tx-index/3000496/1.
[15]https://block-chain.info/it/tx-index/3000536.
[16]https://block-chain.info/it/tx-index/1793329/1.
[17]https://block-chain.info/it/tx-index/3118220/1.
[18]https://block-chain.info/it/tx-index/3126754/0.
[19]https://block-chain.info/it/tx-index/56730867/0.
[20]https://block-chain.info/it/tx-index/60118930.
[21]https://block-chain.info/it/tx-index/629343/2.
[22]https://block-chain.info/it/tx-index/781227.
[23]https://blockchain.info/it/tx-index/134023577/1.

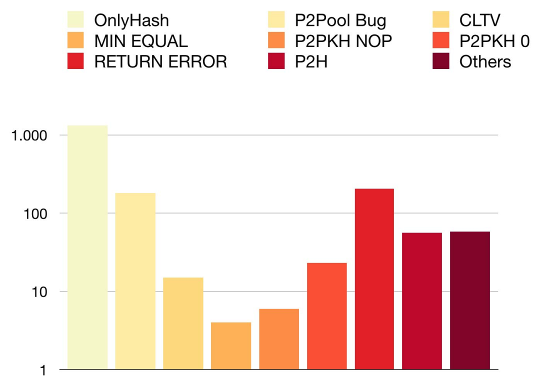

Fig. 2. Distribution of non-standard transactions.

Pay to Hash (P2H) [2012-2015]: These transactions are a simplification of plain P2SH ones; we have a blocking script identical to P2SH, with the difference that the internal hash does not refer to a redeem script, but it is the hash of an hexadecimal string: "OP_HASH160 <HASH160OFSOMETHING> OP_EQUAL". There are two variants, where only the type of hashing operator changes: one corresponds to HASH256, while the other one to SHA256. The two blocking scripts are "OP_HASH256 <HASH256OFSOMETHING> OP_EQUAL" and "OP_SHA256<SHA256OFSOMETHING> OP_EQUAL". The verification step is very simple: the hexadecimal string of the hash in the blocking script is enough to spend the transaction. An example is in the output 0 of a4bfa8ab6435ae5f25dae9d...83ca751f393c1ddc5a837bbc31b[24] and how it is verified in the input 0 of transaction 09f691b2263260e71f363d1d...56a40cc0e4f8c8c2c4a80559b1[25].

Statistics for non-standard transactions: Non-standard transactions are $221,161$ $(0,03\%)$, even if the majority of them, i.e., $219,174$, are UnLocked transactions with a 0 BTC value (all of them in year 2015). This means that they are transactions without blocking scripts, and they do not carry any money: in practice they are "fake" transactions. Thus, if we do not consider such transactions, "real" non-standard ones are only $1,987$: 0.0003% of the total in the block-chain. In Fig. 2 we show the distribution of non-standard transactions. Without considering UnLocked ones, OnlyHash is the most common class, with around a thousand transactions. The second class is the P2Pool Bug, with a few hundreds, while the remaining ones have few outputs. In Fig. 3 we show the percentage of non-standard transactions associated with each miner.

IV. Discussion and Conclusions

In this paper we have presented a report on the statistics concerning standard (seven classes) and non-standard (nine classes) transactions in the Bitcoin block-chain, by considering up to block number $516,040$. The most populated class of transactions is P2PKH; the reason is that they are the default transaction in the Bitcoin client. The second most used class is P2SH, which had a massive growth of over $500,000\%$ transactions in the last 4 years.

[24]https://block-chain.info/it/tx-index/12864193/0.

[25]https://block-chain.info/it/tx-index/12874719.

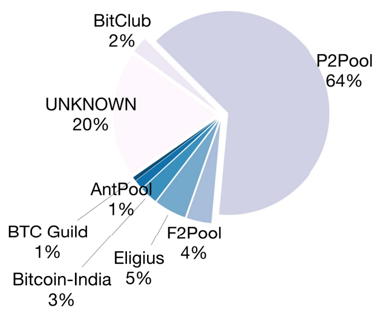

Fig. 3. Distribution of miners in non-standard transactions.

The presented study can help to understand the adherence of the Bitcoin protocol to the intended purposes, by quantifying past and present deviations. As a result we have obtained that only $0,03\%$ $(221,161$ out of $837,854,035)$ of transactions outputs corresponds to non-standard ones: this shows that most of miners and users behave in a standard way.

In the future we plan to study the distribution of non-standard classes along time, and to relate them with amounts of involved bitcoins (also for standard classes). We will also analyse the redeem scripts of P2SH to see what transactions are used inside of them. Finally, we also plan to use analysis and visualization tools [6], [5] to relate transaction types and topologies.

Acknowledgment

This research is supported by project "REMIX" (funded by Banca d'Italia and Fondazione Cassa di Risparmio di Perugia) and projects "ComPAArg" and "Argumentation 360" (funded by "Ricerca di Base 2015–2016", University of Perugia).

References

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 443–458.

[2] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 2014.

[3] M. Bartoletti and L. Pompianu, "An analysis of bitcoin op_return metadata," *CoRR*, vol. abs/1702.01024, 2017.

[4] S. Bistarelli, M. Mantilacci, P. Santancini, and F. Santini, "An end-to-end voting-system based on bitcoin," in *Proceedings of the Symposium on Applied Computing, SAC 2017*, 2017, pp. 1836–1841.

[5] S. Bistarelli, M. Parroccini, and F. Santini, "Visualizing bitcoin flows of ransomware: Wannacry one week later," in *Proceedings of the Second Italian Conference on Cyber Security (ItaSEC)*, ser. CEUR Workshop Proceedings, vol. 2058. CEUR-WS.org, 2018.

[6] S. Bistarelli and F. Santini, "Go with the -bitcoin- flow, with visual analytics," in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES)*. ACM, 2017, pp. 38:1–38:6.

[7] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," http://www.hashcash.org/papers/hashcash.pdf, 2008, [Online; accessed 28-January-2018].

[8] E. D. Rather, D. R. Colburn, and C. H. Moore, "The evolution of forth," in *ACM Sigplan Notices*, vol. 28, no. 3. ACM, 1993, pp. 177–199.