frontiers in Blockchain

ORIGINAL RESEARCH
published: 13 August 2019
doi: 10.3389/fbloc.2019.00007

An Analysis of Non-standard Transactions

Stefano Bistarelli[1]*, *Ivan Mercanti*[2]* and *Francesco Santini*[1]*

[1] Department of Mathematics and Computer Science, University of Perugia, Perugia, Italy, [2] IMT School for Advanced Studies, Lucca, Italy

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, which are resolved by sending the public key and a digital signature created by the corresponding private key. P2PKH transactions are just one among many *standard* classes: a transaction is standard if it passes *Bitcoin Core 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 analyze and classify standard and non-standard transactions, understanding how much the standard behavior is respected.

Keywords: Bitcoin, standard transaction, non-standard transaction, Bitcoin script, P2SH, OP_RETURN

Edited by:
Massimo Bartoletti, University of Cagliari, Italy

Reviewed by:
Roman Matzutt, RWTH Aachen Universität, Germany
Chen Feng, University of British Columbia Okanagan, Canada

***Correspondence:**
Stefano Bistarelli
stefano.bistarelli@unipg.it
Ivan Mercanti
ivan.mercanti@imtlucca.it
Francesco Santini
francesco.santini@unipg.it

Specialty section:
This article was submitted to Non-Financial Blockchain, a section of the journal Frontiers in Blockchain

Received: *15 March 2019*
Accepted: *26 July 2019*
Published: *13 August 2019*

Citation:
Bistarelli S, Mercanti I and Santini F (2019) An Analysis of Non-standard Transactions. Front. Blockchain 2:7. doi: 10.3389/fbloc.2019.00007

1. INTRODUCTION

The white-paper on Bitcoin appeared in November 2008 (Nakamoto, 2008), 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 decentralized, with all the transaction components 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-node software, i.e., *Bitcoin Core*[1]. Our interest is mainly focused on non-standard ones, of which we provide a classification in nine different types, extending some previous analysis for bitcoin[2] (Bistarelli et al., 2018a) and in a manner similar to what done for ethreum (Bistarelli et al., 2019a).

The main motivation behind the paper is to provide an updated and comprehensive screen-shot of standard and non-standard transactions from Bitcoin origins until today. In particular, the goal consists in understanding what and how partial adherence to the Bitcoin protocol or 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. The main result is that only the 0,02% of transactions is non-standard (2009–November 2018). In addition, this study addresses further general questions. For example, there is no particular miner pool that validates only some specific classes of non-standard transactions: all pools that deviate from the standard behavior accept these classes. We also saw that only 2,615 bitcoins (out of more then 17 million in circulation) were definitely "burned" (i.e., made no longer spendable) due to non-standard transactions.

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

The paper, which extends Bistarelli et al. (2018a), is organized as follows: section 2 describes how Bitcoin transactions work, the Bitcoin scripting language, and what the standard transactions are; section 3 shows the non-standard transactions that we found in the block-chain, and related statistics; section 4 shows the statistics of standard and non-classical Bitcoin transactions nested in *P2SH* transactions, focusing on non-standard ones reported in the literature; section 5 classifies OP_RETURN transactions by their byte size, and it also presents related statistics; section 6 shows related works. Finally, section 7 draws the final conclusions and proposes ideas about future work.

2. BACKGROUND

2.1. 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; Antonopoulos, 2017). A private key is a random 256 bit 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 instead describes the destination of bitcoins by providing a challenge to users. Hence, 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."

2.2. UTXO and Memory Pool

A UTXO is an *Unspent Transaction Output* that can be spent as an input in a new transaction. To assemble the candidate block, a Bitcoin miner selects transactions from the memory pool (*mempool* for short): as its name suggests, it is a pool of memorized transactions collected by a miner. The data that is stored in the mempool consists of unconfirmed transactions which still needs to be processed by the Bitcoin Network, by applying a priority metric to each transaction and by adding the highest priority transactions in the next block first than lower priority ones. Today's miners choose which transactions to mine only based on fee-rate, thus prioritizing the transactions with highest fees per kilobyte of transaction size. Any transaction left in the mempool, after the block is filled, will remain in the pool for inclusion in the next block. As transactions remain in the mempool, their inputs "age," as the UTXO they spend get deeper into the block-chain with new blocks added on top. Eventually, a transaction without fees might reach a high enough priority to be included in the block for free (Antonopoulos, 2017).

2.3. Scripting Language

The Bitcoin transactions language *Script* is a Forth-like (Rather et al., 1993) 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, instead the operations can push or pop one or more parameters onto/from the execution stack, operate on them, and possibly push their result back in 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. There are also conditional operators such as OP_EQUAL: it pops two items from the stack and pushes TRUE (represented by number 1) if the 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 (Andrychowicz et al., 2014): any combination of locking and unlocking scripts that result in a final TRUE value is valid. **Figure 1** we show the step-by-step validation procedure of this locking plus unlocking script. We 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, thus removing the two 8 and pushing TRUE as result.

2.4. Opcodes

Opcodes are the operators of the scripting language. Now we describe some operators that we will refer to in the next sections:

1. The OP_HASH160 operator hashes twice the top stack element: first with SHA-256 and then with RIPEMD-160.
2. The OP_CHECKSIG: the entire transaction outputs, inputs, and script are hashed. The signature used by OP_CHECKSIG must be a valid signature for this hash and public key. If it is, 1 is returned, 0 otherwise.
3. The OP_MIN operator returns the smaller of the two top elements into the stack.
4. The OP_DROP operator removes the top stack element.
5. The OP_DEPTH operator puts the number of stack elements onto the stack.
6. The OP_2DUP operator duplicates the two elements on top of the stack.
7. The OP_IF operator executes the statements only if the top stack value is not False. The top stack value is removed.

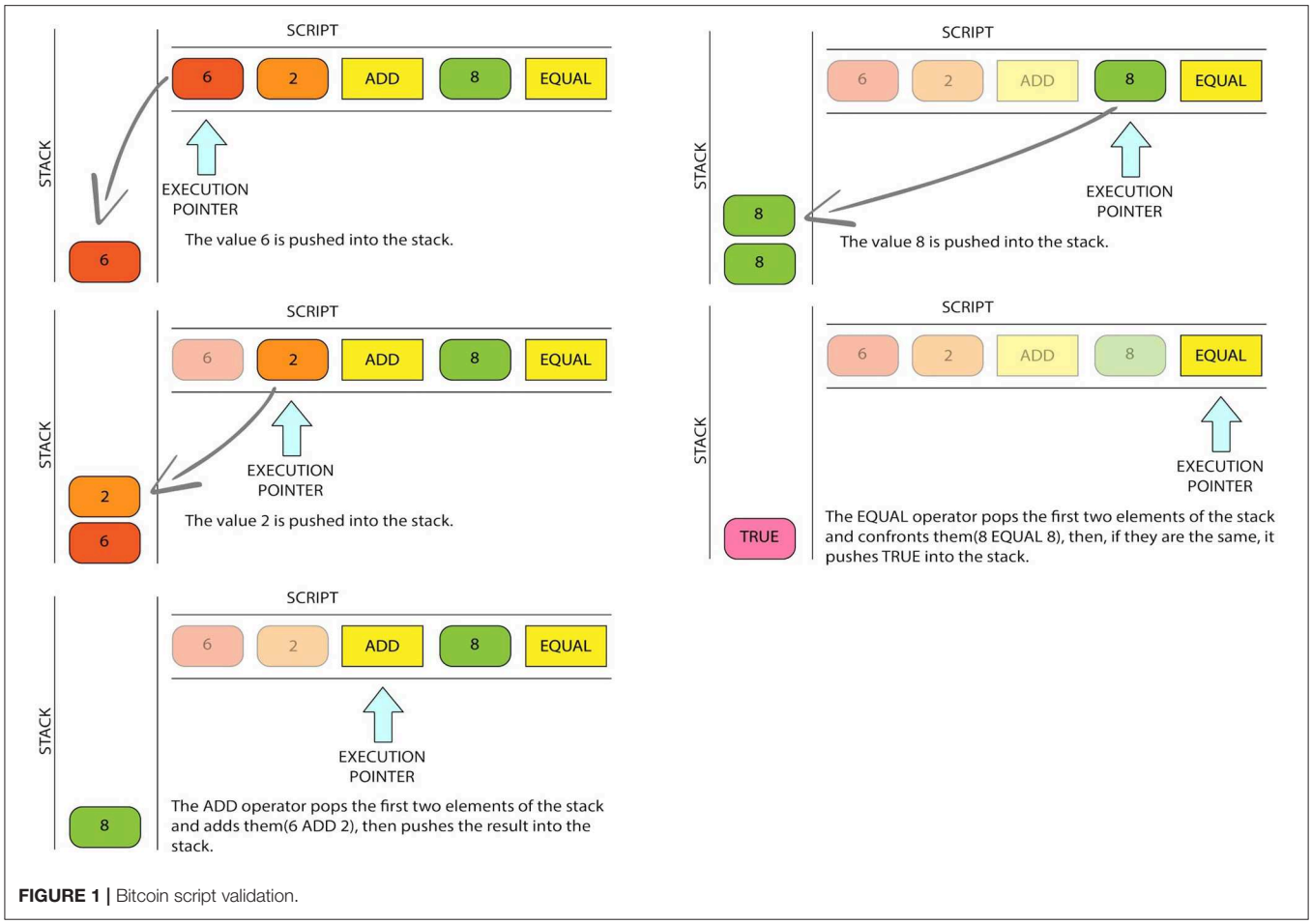

FIGURE 1 | Bitcoin script validation.

8. The OP_ELSE operator executes its statements if the preceding OP_IF or OP_ELSE was not executed.
9. The OP_ENDIF operator ends an if/else block. All blocks must end, or the transaction is invalid. An OP_ENDIF without an OP_IF before is also invalid[3].

2.5. 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]. More accurately, the *isStandard()* function gives TRUE if all the outputs (locking script) use only standard transaction forms. On the other hand, the *isStandardTx()* function gives TRUE if all the inputs (unlocking script) use only standard transaction forms according to the output that they are spending. 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." **Figure 2** shows an example of P2PKH script validation. 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 simpler than P2PKH; it was used in coinbase transactions, i.e., the one with the miners are paid for their job, until July 2012, then they started use the P2PKH (see **Figure 3**). P2PK, as the name suggests, has in its locking script directly the pubkey, instead of its hash: "<PUBLIC KEY A> OP_CHECKSIG." Since 2017 the most used output in a coinbase transaction is the OP_RETURN, which is a provably unspendable (see section 2.5). This coinbase transactions have some P2PKH outputs in order to pay the miners and more OP_RETURN outputs, which do not carry bitcoin (the value of the outputs is 0 BTC). The number

[3]https://en.bitcoin.it/wiki/Script

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

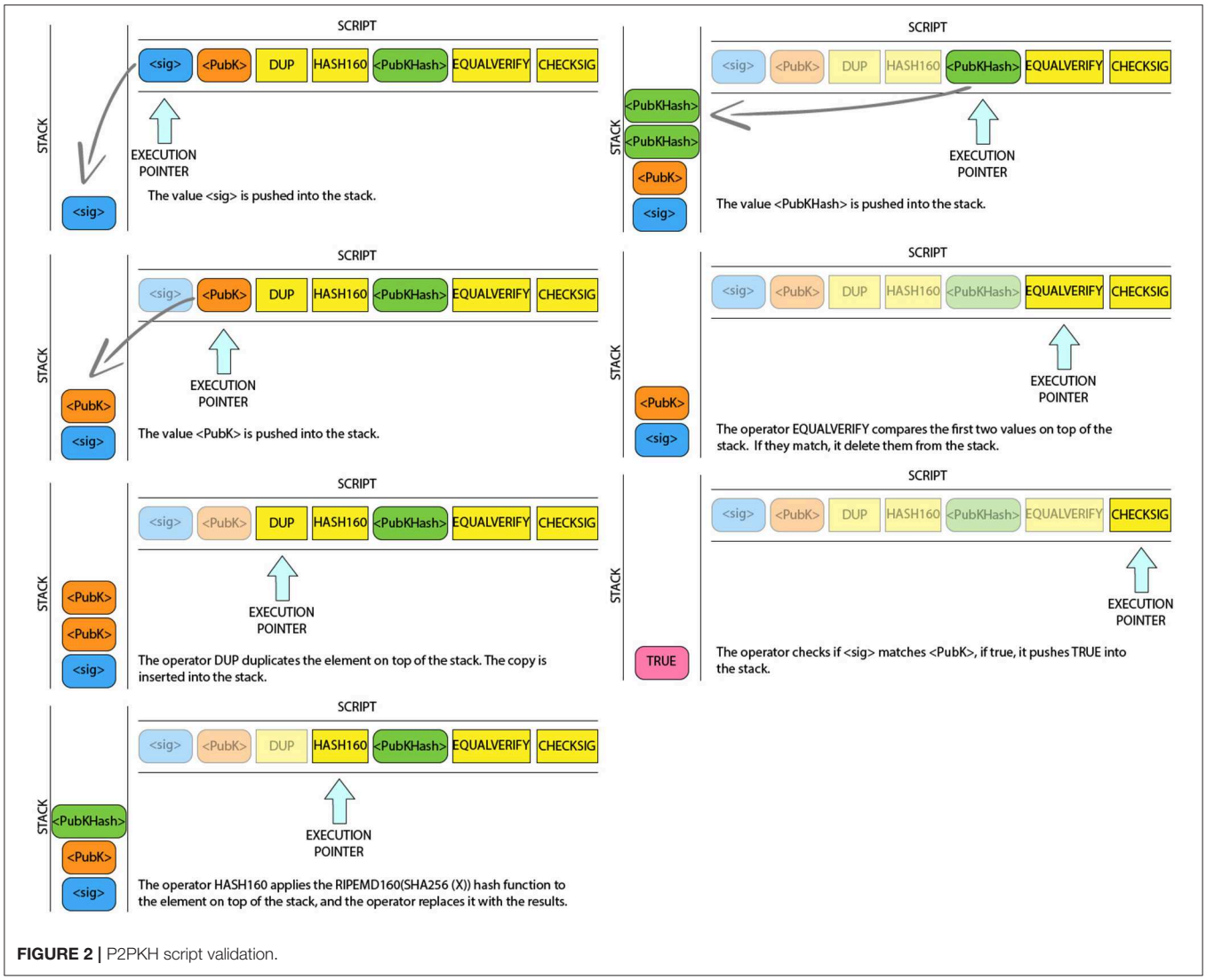

FIGURE 2 | P2PKH script validation.

OP_RETURN outputs Is greater than the number of P2PKH ones. These OP_RETURN outputs are related to Segregated Witness[5] (SegWit) implemented soft fork: They are linked to the Merkle root of the witness tree. The SegWit needs an extended blockheader, but the blockheader cannot be extended without a hardfork, so segwit blocks commit to this witness tree by including the root in an OP_RETURN in the coinbase transaction[6]. **Figure 4** shows the P2PK script validation process. 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 maximum *M* for the current Bitcoin Core implementation is 15. 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." In **Figure 5** the validation steps of this script is visually represented. The locking script can be satisfied with an unlocking script containing: "OP_0 <SIGNATURE 1> <SIGNATURE 2> ... <SIGNATURE M>." Notice that 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.

Data output (OP_RETURN): The *Data output* transactions are used to store data not related to Bitcoin payments. Their form is "OP_RETURN <DATA>." Since any output with OP_RETURN is provably un-spendable. Thus, the output can be immediately pruned from the UTXO[7] set even if it has not been spent. These transactions can be used to save different kinds of

[5]https://en.bitcoin.it/wiki/Segregated_Witness

[6]https://bitcoin.stackexchange.com/questions/74162/op-return-in-a-coinbase-transaction

[7]Unspent Transaction Output, i.e., the transactions that can be spent as an input in a new transaction.

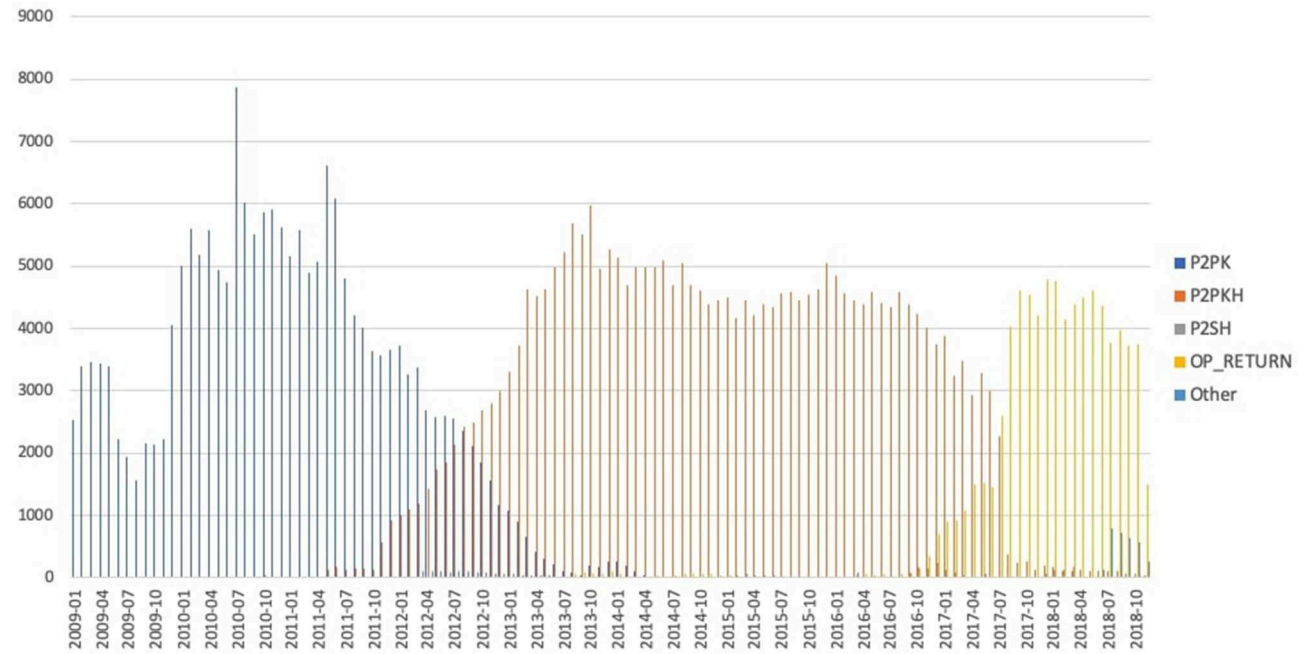

FIGURE 3 | Evolution over time of the output type used in coinbase transactions.

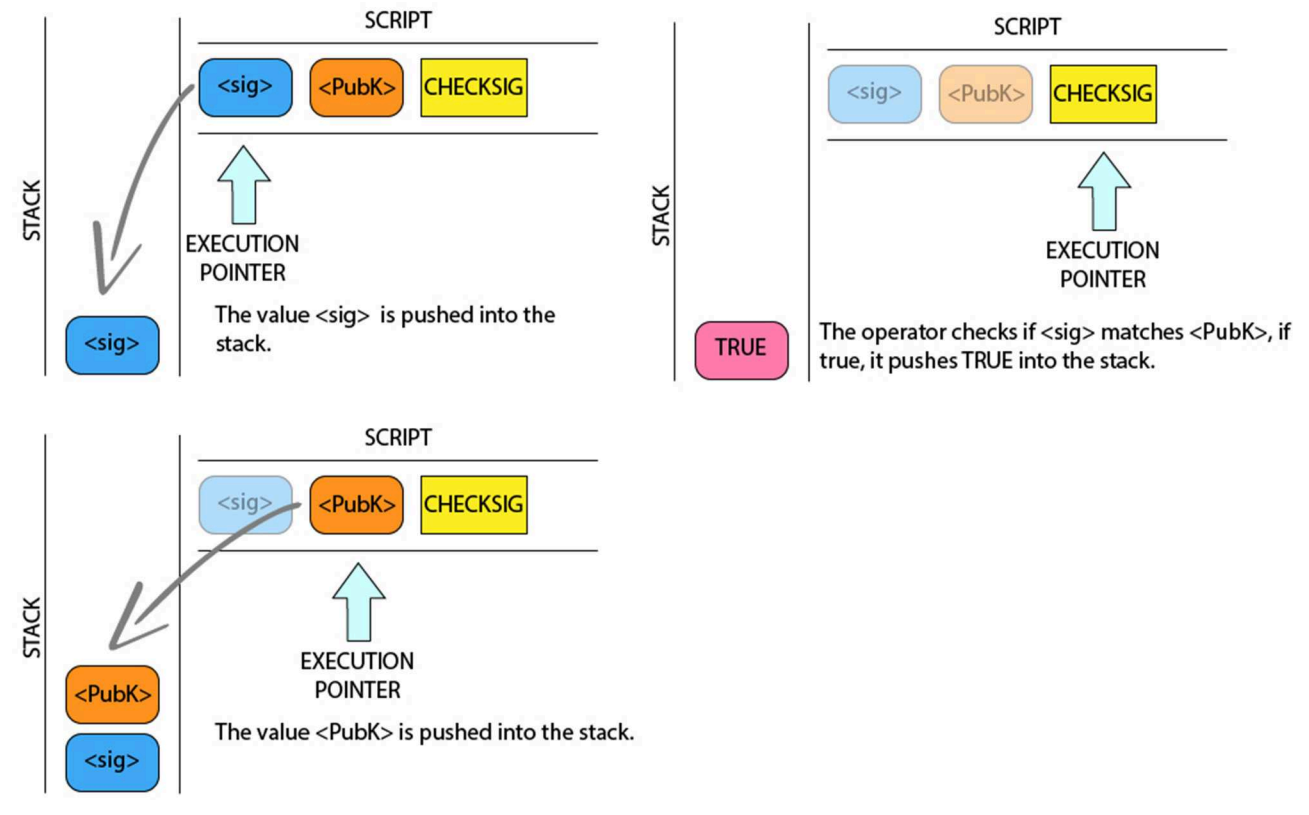

FIGURE 4 | P2PK script validation.

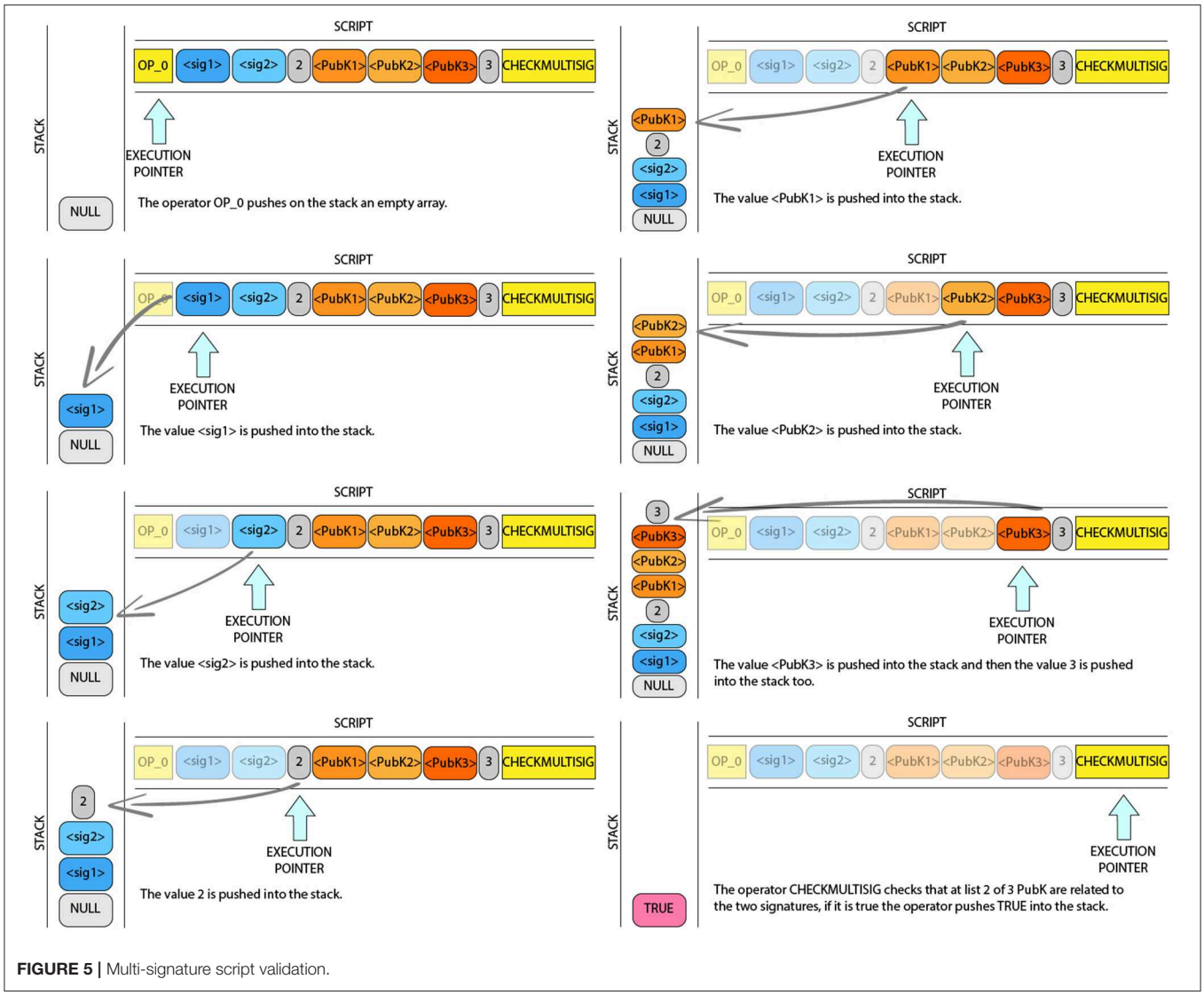

FIGURE 5 | Multi-signature script validation.

information on the block-chain, which is in this way used as an immutable distributed ledger by applications, as e-voting ones (Bistarelli et al., 2017, 2019b), for example. Such transactions are unlockable. Many members of the Bitcoin community believe that use of OP_RETURN is irresponsible in part because Bitcoin was intended to provide a record for financial transactions, not a record for arbitrary data. Additionally, it is trivially obvious that the demand for external, massively-replicated data store is essentially infinite. Despite this, OP_RETURN has the advantage of not creating bogus UTXO entries, compared to some other ways of storing data in the block-chain. This helps miners to be faster in calculating the priority transactions function.

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 (around 180 byte using the compressed form). **Figure 6** shown an example of locking script: "OP_HASH160 <2-OF-5 MULTI-SIGNATURE SCRIPT HASH> OP_EQUAL," with the unlocking script as: "<SIG1> <SIG2> <2-OF-5 MULTI-SIGNATURE SCRIPT>." See **Figure 6** for an example of P2SH script validation. More details about this transaction are given in section 4.

Pay to Witness Public Key Hash (P2WPKH) and Pay to Witness Script Hash (P2WSH): With the introduction of the Segregated Witness[8] (*SegWit*) in Bitcoin, the default transaction P2PKH can be also obtained in a different way. The main differences of the Segregated Witness are the locking script shorter and the signature that are moved outside the unlocking script (see **Figure 7**). In fact, in place of the longer script in P2PKH, the script in P2WPKH is shortened to: "OP_0 <PUBLIC

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

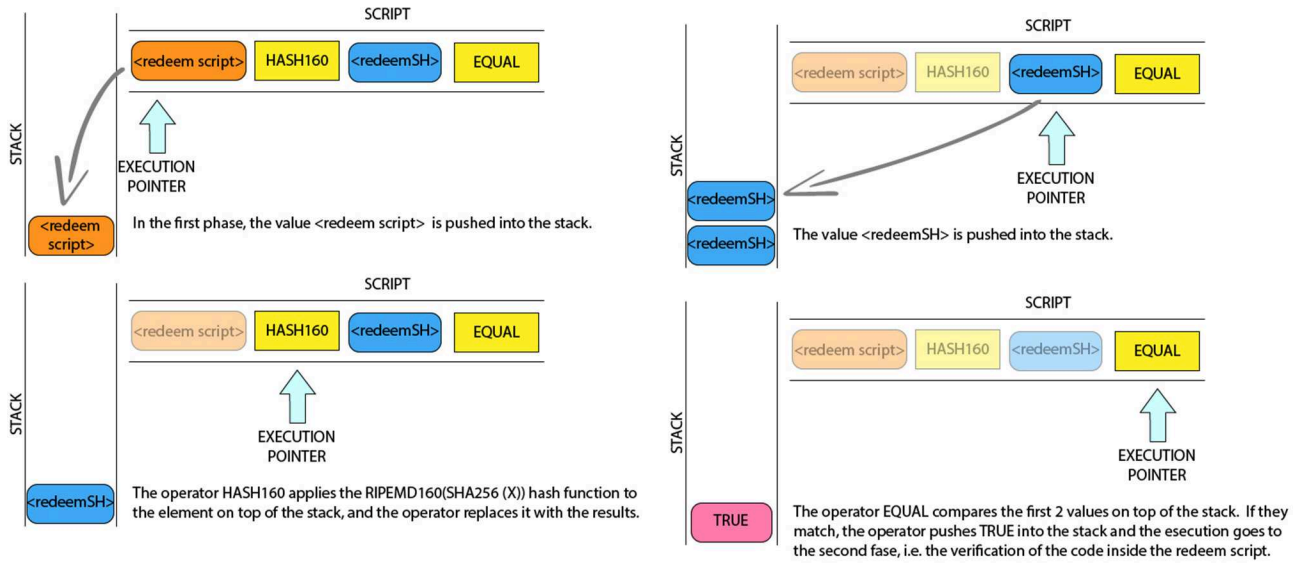

FIGURE 6 | P2SH script validation.

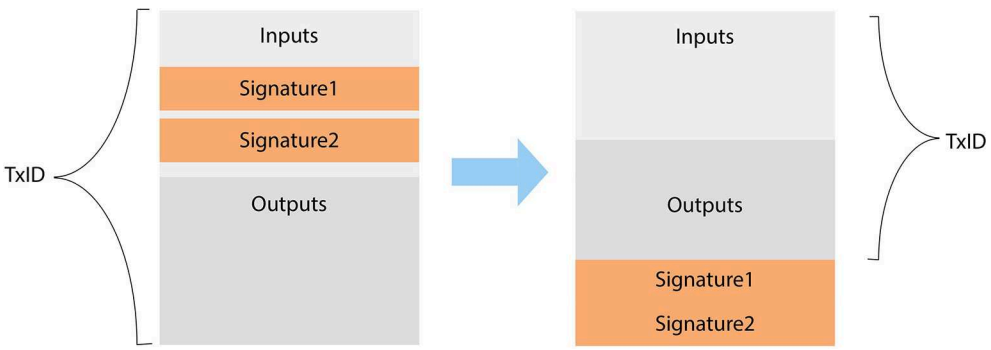

FIGURE 7 | The position of new signatures in witness transactions.

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 components are no longer in the unlocking script, but in the witness field instead.

3. AN ANALYSIS OF BITCOIN TRANSACTIONS

In this section we describe standard and non-standard transactions in the Bitcoin block-chain, by reporting statistics on their number and frequency with the purpose to have a clear view on their "popularity" and acceptance. To accomplish such an analysis we take advantage of a Bitcoin Core node, which we used to fill a PostgreSQL Database[9] in which we have stored all the block-chain blocks up to number 550, 000: until November the 14th 2018. Such a tool is part of the BlockchainVis Suite (Bistarelli et al., 2018b). We consider Bitcoin Core[10] as the reference implementation.

3.1. Statistics for Standard Transactions

Considering the first 550 000 blocks in the block-chain, there are 356 588 805 transactions that generate a total of 968 098 854 outputs, of which 910 274 680 have been spent (94,03%). These include 967 874 499 standard transaction outputs (99,98% of the total number of outputs). Hence, non-standard transaction outputs are only the 0,02% of the total.

In **Figure 8** 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 almost 150 million outputs. Interestingly, the number of P2SH and OP_RETURN transactions

[9]PostGreSQL: https://www.postgresql.org

[10]https://bitcoincore.org

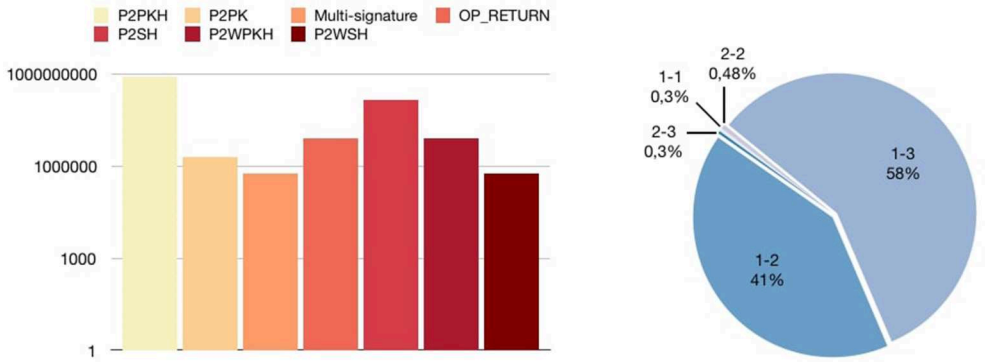

FIGURE 8 | Distribution of standard transactions (left) and distribution of Multi-signature transactions (right).

has considerably increased if compared to the data from early 2018 (Bistarelli et al., 2018a). In **Figure 8** we see that the most used *M-of-N* multi-signature class corresponds to the scheme *1-3*, with 58% of all the multi-signature transactions. The second most used scheme is *1-2*, with 41%. We found also: 38 repetitions of *3-3*, 3 repetitions of *0-1*, only one 1 transaction in the form *3-5*, 1 in *1-9*, and, finally, 1 in the *9-9* form.

3.2. 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 transactions that deviate from the standard enforced by Bitcoin Core can be mined as well: these transactions can be issued in the block-chain thanks to miners that relax these checks enforced by such control functions, as for example Eligius[11]. Non-standard transactions use more complex script forms, represent challenges, or just result from bugs. Their singularity comes from non-standard inputs or outputs.

Correctly validating non-standard transactions can make the creation of future transactions harder for two separate reasons:

1. Some scripts might cause harm to the network.
2. Some scripts might make future upgrades harder.

Concerning the first reason, the non-standard transaction check was first implemented by Nakamoto before P2SH existed, so it could not be so easily circumvented. This gave developers time to better analyze the script language and fix problems with the remaining opcodes: e.g., in some cases[12] it was possible to create a transaction that took 5 h to be verify. However, even if the scripting language is perfectly safe, each script has to be stored by every full node until it is spent as part of the UTXO database. Since a locking script is limited to 10,000 bytes, this means that an attacker can add up to 10 KB to the UTXO set for every output he creates, potentially quickly adding enough data to degrade performance enough that the rate of stale blocks (orphan blocks) mined increases, which would reduce miner profits and encourage them to centralize further to recover that lost revenue.

Concerning the second reason, there are some opcodes the network does not want people to use. These are opcodes that might be redefined in the future, e.g., the OP_NOPX opcodes that have been used for soft forks in the past (OP_NOP1 became OP_CHECKLOCKTIMEVERIFY and OP_NOP2 became OP_CHECKSEQUENCEVERIFY). Lately, Bitcoin Core 0.16.1 quit the use of OP_CODESEPARATOR in non-segwit in preparation for another potential soft fork that will reduce some lingering problems with expensive verification. In those cases, standard transactions forbid both the scriptPubKey and the redeemScript (P2SH) versions (and, when applicable, the segwit P2WSH version), thus the easy circumvention is not possible in that case[13].

One of the reasons to include non-standard transaction in the block-chain could be that miners have a long-term investment in the health of the Bitcoin network. If Bitcoin collapses, then their expensive ASICs are worthless. Miners particularly need bitcoins to remain valuable over the long term because their hardware produces bitcoins over time. If nobody includes transactions in blocks, then bitcoins would be useless and therefore worthless. That would impact on miners long-term investment. If this ever became a problem, transactions would just wind up with higher fees to encourage miners to include them. Right now, enough miners include a transaction with a very small fee and there is no reason in paying more[14].

We searched in the block-chain for these particular transactions and we obtained nine patterns of non-standard transactions. We now describe them by also highlighting the interval of years in which they were confirmed in the block-chain.
Pay to Public Key Hash 0 [2011]: It corresponds to a distortion of P2PKH, with the difference that instead of the hash of a

[11] https://btc.com/stats/pool/Eligius

[12] https://bitslog.com/2017/01/08/a-bitcoin-transaction-that-takes-5-hours-to-verify/

[13] https://bitcoin.stackexchange.com/questions/76541/whats-the-use-of-not-relaying-non-standard-transactions-if-anyone-can-still-use

[14] https://bitcoin.stackexchange.com/questions/11557/what-is-the-motivation-for-miners-to-include-all-recent-transactions-in-a-block

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 [15]. A Bitcointalk thread[16] indicates that this deviation was mainly performed by MtGox. The outputs represent a value of 2609.36304319 BTC, around 8,000$ at that time (around 20 million dollars nowadays).

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." This transaction was probably used to test the OP_NOP operator. It can be unlocked by a script identical to that of P2PKH [17].

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 [18].

P2Pool Bug [2012]: These transactions are due to a bug [19] in the P2Pool[20] mining tool between February 2nd, 2012 and April 1st, 2012. 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 does not make any sense and is not even valid as the OP_IF is not closed by a corresponding OP_ENDIF, hence it is consequently unlockable [21].

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*[22] field of a transaction. In practice, using OP_CHECKLOCKTIMEVERIFY it is possible to make funds provably un-spendable until a certain point in the future, i.e., it is a way to freeze funds up to certain future day [23]. 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 [24].

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." We can see this transaction as a proof of how an equation can be used to generate a bitcoin transaction. This means that anyone can easily unlock such a transaction without any private key [25].

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_EQUALVERIFY." 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." We can consider these transactions as "contest" in the network to find the correct value of the hash in the transactions.

A P2H cannot be considered a *Hash Time-locked Contract* (HTLC). A HTLC is essentially a type of payment in which two people agree to a financial arrangement where one party will pay the other party a certain amount of cryptocurrency. However, the receiving party only has a certain amount of time to accept the payment, otherwise money is returned to the sender. Instead a P2H transaction is different from a HTLC because it generates a payment that can be accepted by the receiver without any time-constraint.

These outputs can only be spent by providing data that once hashed (called *hashlock*) by a cryptographic function is equal to a given hash [26].

The verification step is simple: the hexadecimal string of the hash in the blocking script is enough to spend the transaction [27].

UnLocked (UL) [2015]: This transaction has an empty locking script: 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. Such transactions can be used as a way to donate funds to miners in addition to transaction fees: any miner who mines such a transaction can also include[28] an additional one after sending funds to an address they control [29].

[15] An example is in the first output of the transaction in https://blockchain.info/it/tx-index/1793329/1

[16] https://bitcointalk.org/index.php?topic=50206.0

[17] An example is in output 2 in https://blockchain.info/it/tx-index/629343/2. The unlocking script is in the second input in https://blockchain.info/it/tx-index/781227

[18] See for instance output 0 in https://blockchain.info/it/tx-index/2557393/0

[19] https://bitcointalk.org/index.php?topic=140097.5;imode

[20] http://p2pool.in/

[21] We can see an example in output 1 of https://blockchain.info/it/tx$-$index/2915138/1

[22] https://en.bitcoin.it/wiki/NLockTime

[23] https://en.bitcoin.it/wiki/Script#Freezing_funds_until_a_time_in_the_future

[24] An example of this transaction involves output 1 in https://blockchain.info/it/tx-index/3000496/1, and input 1 in https://blockchain.info/it/tx-index/3000536

[25] An example is in output 1 of the transaction in https://blockchain.info/it/tx-index/3118220/1, and how it has been verified in input 1 in https://blockchain.info/it/tx-index/3126754/0

[26] https://medium.com/@alcio/a-look-at-bitcoin-non-standard-outputs-c97f65cccbb6

[27] An example is in the output 0 in https://blockchain.info/it/tx-index/12864193/0, and how it is verified in the input 0 of the transaction in https://blockchain.info/it/tx-index/12874719

[28] https://en.bitcoin.it/wiki/Script#Anyone-Can-Spend_Outputs

[29] An example is in output 0 in https://blockchain.info/it/tx-index/56730867/0. How it is spent is instead represented by input 0 in https://blockchain.info/it/tx-index/60118930

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 opcode higher than what is really in the code itself. For example an OP RETURN script could ask to insert in the stack the next 40 bytes in the code, but if there are only 28 bytes, the execution fails. This transaction is apparently due to a programming error. The locking script is: "OP_RETURN ERROR" (an error is returned)[30].

OP_2 OP_3 ERROR [2017-2018]: These transactions are similar to OP_RETURN ERROR, but they have no OP_RETURN in the script code. As the OP_RETURN ERROR, their code asks to push onto the stack a number of bytes higher than what is really in the code itself. Probably these transactions, like the OP_RETURN ERROR, are related to an implementation error This is the locking script: "OP_2 OP_3 ERROR" (an error is returned)[31].

Statistics for non-standard transactions: Non-standard transactions are 224 355 (0,02%), 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 5 181: 0,000 5% of the total number in the block-chain.

In **Figure 9** we show the distribution of non-standard transactions. Without considering unlocked ones, 2 3 ERROR is the most common class, with more than three thousand transactions. The second class is the OnlyHash, with almost one thousand outputs, while the remaining ones have few outputs. In **Figure 9** we show the percentage of non-standard transactions associated with each miner. In order to identify the miner of a transaction we looked for the block of this transaction. Then we took the coinbase transaction of the block, in this particular transaction there is a field called just coinbase where the miners put their id. We classified the miners according to these tags [32].

In **Figure 10** we associate the percentage of non-standard transactions: we found that year 2018 has more than 3,200 occurrences of this kind of transactions, all of type OP_2 OP_3 ERROR. Overall, these non-standard transactions contain almost 2,615 bitcoins that are no longer spendable.

4. PAY TO SCRIPT HASH

As we introduced before, the P2SH transactions contain the hash of a script (called *redeem script*) in their locking script. In the block-chain there are 149 410 668 P2SH transactions of which 140 620 401 are spent (94,12%). Hence, we decided to analyse the content of the redeem script inside the unlocking script of the spent P2SH transactions. In the following of this section we show the results we obtained.

[30]An example of it is in output 1 in https://blockchain.info/it/tx-index/134023577/1

[31] An example is represented in output 2 in https://www.blockchain.com/btc/tx/c2cf3a1c3752304803661121edd044bbce7d70e2a43d73a46302ea5c5a868f16

[32]https://github.com/blockchain/Blockchain-Known-Pools/blob/master/pools.json

4.1. Standard Transactions in P2SH

Like in the block-chain analysis, also inside P2SH transactions the majority of the transactions is standard. In fact there are 140 509 279 standard transactions, which are 99,92% of the total. In **Figure 11** we show the distribution of standard transactions inside P2SH ones. The most frequent class of transactions is not the P2PKH (only 447), as in section 3, but it is the multi-signature one, with more than 90 million occurrences (65,7%). The second one is the P2WKH with almost 35 million transactions (24,6%).

4.2. Non-standard Transactions in P2SH

Inside P2SH we found 111 122 non standard transactions (0,08% of the total); this amount is four times more than what we obtained when we "simply" analyzed the block-chain (0,02%). We found several new classes of non-standard transactions, which are different from previous ones.

4.2.1. OP_CHECKLOCKTIMEVERIFY OP_DROP (CLVT)

We found five different types of transactions that take advantage of the CLVT operator, that, as we have already seen in section 2.5, makes transaction provably unspendable until a certain date. Essentially, it allows users to create a Bitcoin transaction of which the outputs are spendable only at some point in the future. CLTV is necessary for properly functional payment channels (e.g., lightning network). These channels are effectively a series of "off-chain" transactions, that benefit from all the security of typical on-chain transactions, and with some added benefits [33].

One of them is equal to a group of transactions already present in the block-chain: "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP." The second is a P2PK locked with the CLVT operator, and this is the locking script: "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP <PUBLIC KEY A> OP_CHECKSIG." This script means it is not possible to use the signature (related to the public key) to spend this transaction before the date in the script. The third one is a P2PKH locked with a CLVT operator, this is the script: "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG." Then we have the class "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP 1 OP_ADD 2 OP_EQUAL," which simply needs the number corresponding to the equation $x + 1 = 2 \wedge x = 1$: it consequently waits until the date has expired. The last one is a P2PKH variant that also needs a further hash to be unlocked; the locking script is "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP OP_SHA256 <SHA256OFSOMETHING> OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG." As we can see in **Figure 11**, the most frequently type adopted in transaction is the P2PK one, with almost 1 500 outputs.

[33]https://bitcoinmagazine.com/articles/checklocktimeverify-or-how-a-time-lock-patch-will-boost-bitcoin-s-potential-1446658530

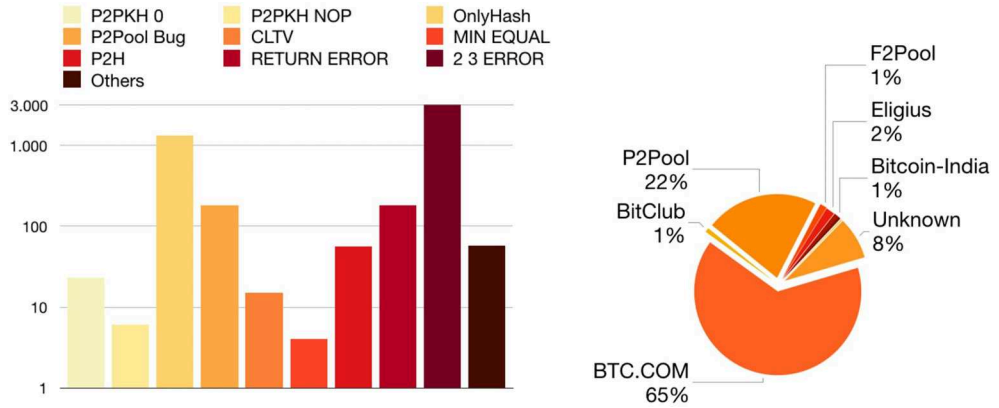

FIGURE 9 | Distribution of non-standard transactions **(left)** and distribution of their miners **(right)**.

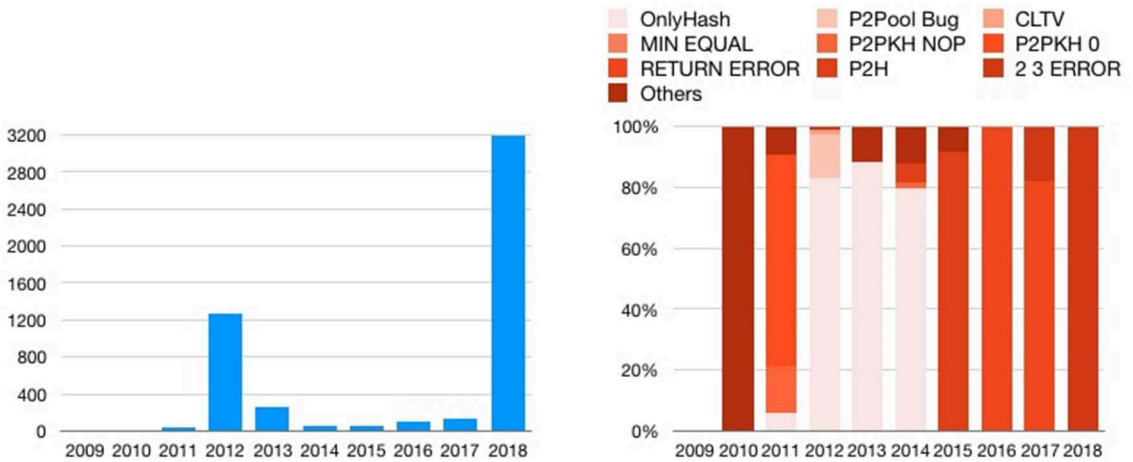

FIGURE 10 | Distribution of non-standard transactions over time **(left)** and distribution of non-standard transactions type over time **(right)**.

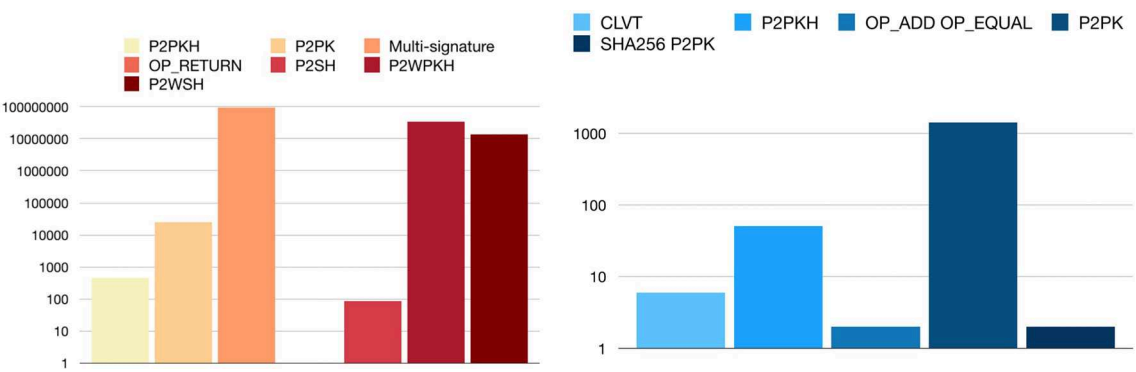

FIGURE 11 | Distribution of standard **(left)** and CLVT transactions inside P2SH **(right)**.

4.2.2. OP_DROP

This transaction allows for storing some data in block-chain without making the transaction unspendable. In fact, all these transaction start with <DATA> OP_DROP where <DATA> is what we want to put in block-chain and OP_DROP is the operator that removes the data from the stack in order to make the execution same as of a standard transaction.

We found four different types of transactions that use OP_DROP. The first type does not need anything to be unlocked, that is the script is: "<DATA> OP_DROP 1." Then there is the $2-2$ multi-signature type "<DATA> OP_DROP 2 <PUBLIC KEY A> <PUBLIC KEY B> 2 OP_CHECKMULTISIG," which needs the two signatures to be unlocked, like the normal $2-2$ multi-signature. The third one is a P2PKH with OP_DROP, identified by the "<DATA> OP_DROP OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG." Also this one needs only the signature, as for P2PKH transactions. The last one is an OP_DROP with a P2PK: "<DATA>OP_DROP <PUBLIC KEY A> OP_CHECKSIG." It can be unlocked as a classical P2PK transaction.

In **Figure 12** we can see that the most used type is the $2-2$ multi-signature one, with almost 25 000 occurrences. We can say that these transactions are just standard outputs in disguise, using the OP_DROP operator to add data that is discarded during verification [34].

4.2.3. OP_Hash160 OP_Equalverify

We found four different types of transactions that start with OP_HASH160 OP_EQUALVERIFY. The first one has this locking script: "OP_HASH160 <HASH160OFSOMETHING>OP_EQUAL 1" or "OP_HASH160 <HASH160OFSOMETHING> OP_EQUAL 0 1." They could be transactions made to have a P2SH that can be unlocked by revealing only the redeem script, in fact these scripts do not need anything to be unlocked because at the end they push 1 in the stack; hence, the transaction is always verified.

Then there is the P2PK that start with a series of OP_HASH160 OP_EQUALVERIFY, with the locking script: "(OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY)*N <PUBLIC KEY A> OP_CHECKSIG"[35]. To unlock the script, it is needed to know all the strings of the hashes and the private key corresponding to the public key in the script. We can consider this transaction like a challenge to a specific person (the owner of the private key corresponding to the public key in the script): when he found all the strings of the hashes in the script, he can take the money.

There is also a variant with this script: "(OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY)*N <PUBLIC KEY A> OP_CHECKSIGVERIFY <DATA> OP_DROP OP_DEPTH 0 OP_EQUAL," that can be unlocked like the previous one because the new part checks that there is no element in the stack and it is possible only if all the given strings and the signature are correct. Similar to the previous one, this transaction is a challenge to a specific person, which can take the money only if he knows all the strings of the hashes in the script, but in addition there is the <DATA> OP_DROP sequence, as we saw in section 4.2.2, allows for storing some data in block-chain without making the transaction unspendable.

The last one is only with OP_HASH160 OP_EQUALVERIFY, whose identifying script is "(OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY)*N OP_HASH160 <HASH160OFSOMETHING>OP_EQUAL." To unlock it we need to know all the hashing strings. Also this transaction could be a challenge: the first user who finds all the strings of the hashes, can take the money.

In **Figure 12** we show the distribution of OP_HASH160 OP_EQUALVERIFY transactions. The one with OP_DEPTH is the most common class, with more than 6 500 transactions extracted from the block-chain.

4.2.4. OP_IF

These transactions are characterized by the presence of the OP_IF. In fact this operator, as in C or JAVA, generates different execution branches, and the receiver can chose the one he prefers.

We found seven different classes of transactions that start with OP_IF, each of them characterized by the following scripts:

1. "OP_IF
   OP_SIZE 32 OP_EQUALVERIFY OP_SHA256
   <SHA256OFSOMETHING>OP_EQUALVERIFY OP_DUP
   OP_HASH160 <PUBLIC KEY A HASH>
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP
   OP_DUP OP_HASH160 <PUBLIC KEY B HASH>
   OP_ENDIF
   OP_EQUAL OP_CHECKSIG".
   This transaction can be unlocked in two ways: the first with a 32-byte string obtained from the SHA256 hash and the signature of A; the second one, after the date in the script, only with the signature of B. We can see this transaction like a challenge between A and B (the owner of the private keys corresponding to the public keys A and B in the script): if A finds the 32-byte string before the date in the script can take the money, otherwise B will take the bitcoins.
2. There is also another type equal to the last one but without the size check:
   "OP_IF
   OP_SHA256 <SHA256OFSOMETHING>OP_EQUALVERIFY
   OP_DUP
   OP_HASH160 <PUBLIC KEY A HASH>
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP
   OP_DUP OP_HASH160 <PUBLIC KEY B HASH>
   OP_ENDIF
   OP_EQUAL OP_CHECKSIG".
   This transaction presents the same challenge as in the previous one, but the string's length is not fixed, so A has to find the string before the date in the script or B will take the money.
3. We found also a version with the branch reversed and using Hash160 instead of SHA256:
   "OP_IF
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP
   <PUBLIC KEY A> OP_CHECKSIG
   OP_ELSE
   OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY

[34] https://medium.com/@alcio/a-look-at-bitcoin-non-standard-outputs-c97f65cccbb6

[35] In this script, N is a integer number greater than 0 and it represents the number of times that a particular part of code is repeated.

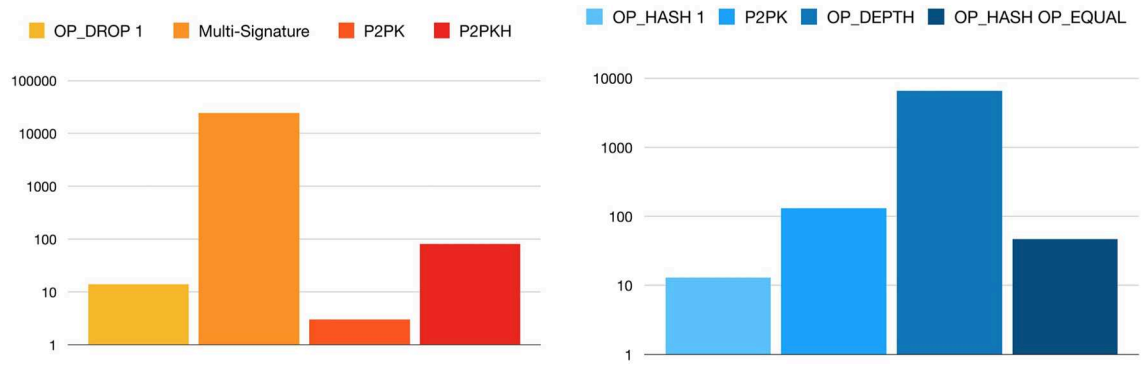

FIGURE 12 | Distribution of OP_DROP **(left)** and OP_HASH160 OP_EQUALVERIFY transactions inside P2SH **(right)**.

<PUBLIC KEY B> OP_CHECKSIG
OP_ENDIF".
This is identical to the second one but with the branch inverted, i.e., now B has to find the strings or A will take the money.

4. One more variant, which needs 15 original strings, is this:
"OP_IF
(OP_RIPEMD160 <RIPEMD160OFSOMETHING>
OP_EQUALVERIFY)*15 <PUBLIC KEY A> OP_CHECKSIG
OP_ELSE
<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP
<PUBLIC KEY B> OP_CHECKSIG
OP_ENDIF".
Also this transaction can be considered like a challenge very similar to the second one, but A has to find 15 strings of the hashes before the date in the script, otherwise B will take the money.
5. A class that can be spent immediately with two signatures or, after the date inside the script, with one signature only:
"OP_IF
<PUBLIC KEY A> OP_CHECKSIGVERIFY
OP_ELSE
<DATA>OP_CHECKLOCKTIMEVERIFY OP_DROP
OP_ENDIF
<PUBLIC KEY B> OP_CHECKSIG".
This could be consider a transaction between two people (A and B) who do not trust each other, in fact if everything is well and good and A does not disappear, they can take the bitcoins, otherwise after the date in the script B can take the bitcoins.
6. A class with 2-2 multi-signatures and CLVT:
"OP_IF
2 <PUBLIC KEY A> <PUBLIC KEY B> 2
OP_CHECKMULTISIG
OP_ELSE
<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP
<PUBLIC KEY A> OP_CHECKSIG
OP_ENDIF".
This transaction can be considered exactly like the previous one, in fact the sequence 2 <PUBLIC KEY A> <PUBLIC KEY B> 2 OP_CHECKMULTISIG is identical to <PUBLIC KEY A> OP_CHECKSIGVERIFY <PUBLIC KEY B> OP_CHECKSIG.
7. The last class is represented by transactions that do not need anything to be unlocked: at the end the script pushes 1 in the stack. For example, the script is
"OP_IF
<DATA> 15 <PUBLIC KEY A> OP_CHECKMULTISIG
OP_ENDIF
1".
They could be transactions prepared to have a P2SH that can be unlocked by revealing only the redeem script, in order to make easier the unlock.

As we can see in **Figure 13**, the most used transaction is the OP_IF 1, with more than 70 000 results.

4.2.5. OP_RIGHT

This type of transaction has a script that contains only "OP_RIGHT." This operator[36] takes a string and a position and it pushes only the characters on the right w.r.t. that position in the string. To unlock this script, it is enough to assemble an unlocking script with a number and a string in a way that the result of the OP_RIGHT is different from 0. Like the OP_HASH 1 or the OP_IF 1 in the previous sections this transaction could be used to make a P2SH that can be unlocked by revealing only the redeem script.

4.2.6. OP_2DUP Multi-signature

These transactions are similar to the multi-signature transactions, but with some noticeable differences: "OP_2DUP OP_EQUAL OP_NOT OP_VERIFY 2 <PUBLIC KEY A> OP_DUP 2 OP_CHECKMULTISIG." To unlock this script, two signatures from the same private key are needed. The reason is that the first operator OP_2DUP duplicates the two signatures, and then OP_EQUAL checks if they are the same; however, they are different, and then it pushes 0 in the stack. Now, OP_NOT changes 0 in 1 and OP_VERIFY removes 1 from the stack. At the end, the presented scheme corresponds to a plain multi-signature.

[36]https://en.bitcoin.it/wiki/Script

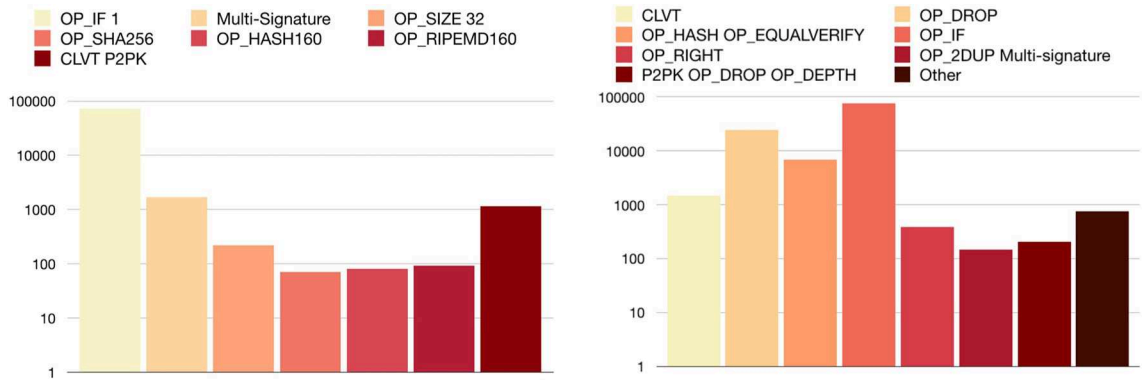

FIGURE 13 | Distribution of OP_IF(left) and non-standard transactions inside P2SH (right).

This transaction is very dangerous for the receiver, in fact it requires two signature from the same private key. This exposes the private key to the risk of being recovered from the public key, i.e., the risk that anyone can have this private key[37].

4.2.7. P2PK OP_DROP OP_DEPTH

This transaction looks like a P2PK but at the end of the script it checks whether the stack is empty, this is the script: "<PUBLIC KEY A> OP_CHECKSIGVERIFY <DATA> OP_DROP OP OP_DEPTH 0 OP_EQUAL." So to unlock this script, only the signature generated by the private key of A is needed. Like the OP_DROP (see section 4.2.2), this transaction allows for storing some data in block-chain without making the transaction unspendable.

In **Figure 13** we show the distribution of non-standard transactions inside P2SH transactions. The most used is the OP_IF class, with almost 80 000 outputs. The second one, with almost 25 000 occurrences, is the OP_DROP transaction.

5. OP_RETURN

We analyze the nulldata transaction outputs, also called OP_RETURN, and in this way we update with new data the work by Bartoletti et al. (2019). As introduced before, this is a particular kind of transaction, since it cannot be spent and it is used only to store data in the block-chain, using it like an immutable distributed ledger. In this case, our goal is to study the use of OP_RETURN over the year.

5.1. Data Dimension

The data dimension inside the OP_RETURN transaction is different also because the standard dimension changed during years. In fact, in the Bitcoin Core 0.9.0[38] the limit was only 40 bytes. Then from the 0.10.0[39] release on February 16th 2015, Bitcoin nodes could choose whether to accept or not OP_RETURN transactions, and to set a maximum dimension. The 0.11.0[40] release on July 12th 2015 extended the data limit to 80 bytes. Finally, the 0.12.0[41] release on February 23th 2015 set the maximum to 83 bytes (80 byte default, plus 3 bytes overhead).

In **Figure 14** we show the distribution of OP_RETURN data size. The most used size is 20 bytes with more than 4 million occurrences, the second one is 80 bytes with almost 1 million outputs, and the third is the 40 bytes with more than 500 thousands occurrences. There are also other transactions with only one occurrence, which we do not show in **Figure 14**, exactly with size 95, 114, 134, 190, 449, and 983 bytes. There are also 296 518 output with data size 0, i.e., empty, of which 222 348 were made in September 2015 by compromised addresses[42]. These transactions were definitely used as stress tests for the Bitcoin network (Baqer et al., 2016).

In **Figure 15** we show the distribution of OP_RETURN transactions over time. It seems clear that every year the OP_RETURN occurrences double, and this proves that their use is increasing at a high rate.

5.1.1. "Non-standard" OP_RETURN

We analyzed transactions according to the aforementioned Bitcoin Core update, with the purpose to find those that did not respect size rules, thus we can call that "non-standard." We found 797 results, of which 784 were registered before the release of Bitcoin Core 0.9.0 (on average they have a size of 38 bytes) and 13 registered during the 0.9.0 release, but with more than 40 bytes (on average they have a size of 61 bytes). In **Figure 15** we show the distribution of miners that accepted transactions with "non-standard" OP_RETURN transactions. P2Pool is the miner pool that mined the largest number of them.

The last analysis that we did is the amount of Bitcoin "lost" in these transactions. We see that only 57 038 (0,68% of all OP_RETURN transactions) spent bitcoins with a total amount of 3,715 725 52 BTC. The maximum spent for a transaction is 0,018 454 BTC, the minimum is 1 Satoshi (10^{-8} BTC).

[37] https://allprivatekeys.com/random-vulnerability.php
[38] https://bitcoin.org/en/release/v0.9.0
[39] https://bitcoin.org/en/release/v0.10.0
[40] https://bitcoin.org/en/release/v0.11.0
[41] https://bitcoin.org/en/release/v0.12.0
[42] https://allprivatekeys.com/random-vulnerability.php

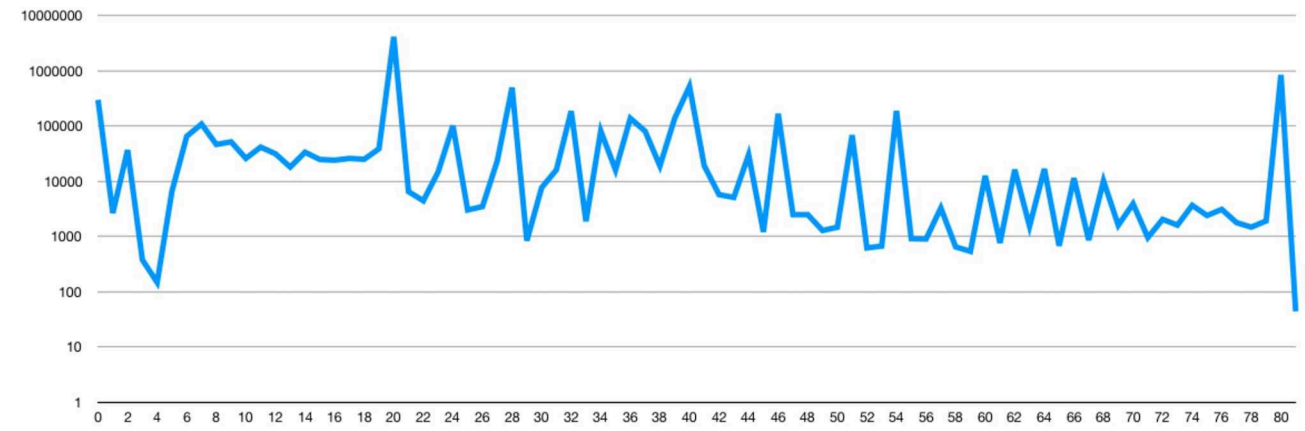

FIGURE 14 | Distribution of OP_RETURN transactions divided by size.

FIGURE 15 | Distribution of OP_RETURN transactions over time **(left)** and Distribution of miners in "non-standard" OP_RETURN transactions **(right)**.

6. RELATED WORK

Analyzing and understanding the Bitcoin block-chain is as complicated (due to the amount of data) as interesting. Several analysis can be found in the literature.

In 2014 Ken Shirriff's blog[43] studied some methods for inserting arbitrary data into Bitcoin block-chain and also what kind of data can be (or is already) stored.

A few months later QuantaBytes[44] surveyed Bitcoin transactions in block-chain and found three classes of non-standard transaction.

In 2018 Sward et al. (2018) improved the study on inserting arbitrary data into Bitcoin's block-chain.

Then Matzutt et al. (2018a) describes the problem of inserting harmful content into a block-chain; in particular they propose conceptual countermeasures to heuristically reject transactions holding unintended content with high probability. They find that mandatory minimum fees and mitigation of transaction manipulability significantly raise the bar for inserting malicious content into a block-chain.

In the same period, Matzutt et al. (2018b) the authors provide a systematic analysis of the benefits and threats of arbitrary block-chain content. They show that certain illegal content can render the mere possession of a block-chain illegal. Their analysis reveals more than 1,600 files on the block-chain, e.g., links to child pornography. This analysis highlights the importance for future block-chain to be designed to address the possibility of unintended data insertion and protect users.

In 2019 Bartoletti et al. (2019) empirical study the usage of OP_RETURN over the years and they identify several protocols based on OP_RETURN, which they classify by the application domain and their space consumption.

7. 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 550 000, i.e., until November 14th 2018.

[43] http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html
[44] https://www.quantabytes.com/articles/a-survey-of-bitcoin-transaction-types

The most populated class of transactions is P2PKH; the reason is that they represent the default transaction in Bitcoin clients. The second most used class is P2SH, which had a massive growth of over 40% transactions from Bistarelli et al. (2018a).

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 0,02% (224 355 out of 968 098 854) of transactions outputs corresponds to non-standard ones: this shows that most of miners and users behave in a standard way. We noticed that only the 0,015% (2 615 out of more than 17 million) of all the circulating bitcoins was burned by non-standard transactions. We saw that the most used transaction inside P2SH transactions is the multi-signature one. We also show that the most used size in bytes of an OP_RETURN transaction is 20 bytes.

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 analyze OnlyHash transactions, which we surveyed in section 2.5. The aim is to see if they could be treated as "colored coin" transactions. Finally, we also plan to use analysis and visualization tools (Bistarelli and Santini, 2017; Bistarelli et al., 2018c) to relate transaction types and topologies together.

DATA AVAILABILITY

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct and intellectual contribution to the work, and approved it for publication.

FUNDING

This work was supported by project Agrichain (funded by Fondazione Cassa di Risparmio di Perugia 2018-2020), project ASIA (funded by INDAM-GNCS) and RACRA18 (Knowledge Representation and Automated Reasoning 2018) Fondi ricerca di Base 2018 (Mathematics and Computer Science Department).

ACKNOWLEDGMENTS

The paper extends Bistarelli et al. (2018a) published in Crypto Valley Conference on Blockchain Technology CVCBT 2018, Zug, Switzerland.

REFERENCES

Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. (2014). "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy* (Berkeley, CA), 443–458.

Antonopoulos, A. M. (2017). *Mastering Bitcoin: Programming the Open Blockchain, 2nd Edn.* Champaign, IL: O'Reilly Media, Inc.

Baqer, K., Huang, D. Y., McCoy, D., and Weaver, N. (2016). "Stressing out: Bitcoin "stress testing"," in *Financial Cryptography and Data Security*, eds J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff (Berlin; Heidelberg: Springer), 3–18.

Bartoletti, M., Bellomy, B., and Pompianu, L. (2019). A journey into bitcoin metadata. *J. Grid Comput.* 17, 3–22. doi: 10.1007/s10723-019-09473-3

Bistarelli, S., Mantilacci, M., Santancini, P., and Santini, F. (2017). "An end-to-end voting-system based on bitcoin," in *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017,* eds A. Seffah, B. Penzenstadler, C. Alves, and X. Peng (Marrakesh: ACM), 1836–1841.

Bistarelli, S., Mazzante, G., Micheletti, M., Mostarda, L., and Tiezzi, F. (2019a). "Analysis of ethereum smart contracts and opcodes," in *Advanced Information Networking and Applications - Proceedings of the 33rd International Conference on Advanced Information Networking and Applications, AINA 2019, Matsue, Japan, March 27-29, 2019, Vol. 926 of Advances in Intelligent Systems and Computing,* eds L. Barolli, M. Takizawa, F. Xhafa, and T. Enokido (Matsue: Springer), 546–558.

Bistarelli, S., Mercanti, I., Santancini, P., and Santini, F. (2019b). End-to-end voting with non-permissioned and permissioned ledgers. *J. Grid Comput.* 17, 97–118. doi: 10.1007/s10723-019-09478-y

Bistarelli, S., Mercanti, I., and Santini, F. (2018a). "An analysis of non-standard bitcoin transactions," in *Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018* (Zug: IEEE), 93–96.

Bistarelli, S., Mercanti, I., and Santini, F. (2018b). "A suite of tools for the forensic analysis of bitcoin transactions: preliminary report," in *Euro-Par 2018: Parallel Processing Workshops - Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018* (Turin: Springer), 329–341.

Bistarelli, S., Parroccini, M., and Santini, F. (2018c). "Visualizing bitcoin flows of ransomware: Wannacry one week later," in *Proceedings of the Second Italian Conference on Cyber Security (ItaSEC)*, volume 2058 of *CEUR Workshop Proceedings*, 33. Available online at: CEUR-WS.org.

Bistarelli, S., and Santini, F. (2017). "Go with the -bitcoin- flow, with visual analytics," in *Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017* (Reggio Calabria: ACM), 38:1–38:6.

Matzutt, R., Henze, M., Ziegeldorf, J. H., Hiller, J., and Wehrle, K. (2018a). "Thwarting unwanted blockchain content insertion," in 2018 *IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20* (Orlando, FL: IEEE Computer Society), 364–370.

Matzutt, R., Hiller, J., Henze, M., Ziegeldorf, J. H., Müllmann, D., Hohlfeld, O., et al. (2018b). "A quantitative analysis of the impact of arbitrary blockchain content on bitcoin," in *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)* (Nieuwpoort: Springer).

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*.

Rather, E. D., Colburn, D. R., and Moore, C. H. (1993). "The evolution of forth," in *History of Pro-gramming Languages Conference (HOPL-II), Preprints, Cambridge, Massachusetts, USA, April 20-23, 1993,* eds J. A. N. Lee and J. E. Sammet (Cambridge, MA: ACM), 177–199.

Sward, A., Vecna, I., and Stonedahl, F. (2018). Data insertion in bitcoin's blockchain. *Ledger*, 3.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.