Efficient and Reliable Service Detection on Bitcoin

Vincent Jacquot*, Nada Hammad‡, Benoit Donnet*
*Université de Liège, Montefiore Institute, Belgium
‡TRM Labs

***Abstract*—The rise of cryptocurrencies has created new avenues for criminal money exchanges. Among various techniques, Bitcoin address clustering plays a crucial role in detecting and grouping addresses owned by the same entity. This fundamental step is essential for deanonymizing addresses and analyzing the flow of funds in the blockchain. This advancement contributes to the battle against illicit commerce, money laundering, fraud, scams, and similar activities.**

In this paper, we introduce two new heuristics, NSS and PEKET. NSS leverages Bitcoin non-standard scripts, while PEKET exploits the re-use of public keys to establish connections controlled by the same entity. Our contributions encompass (*i*) the detailed explanation of these two novel methods; (*ii*) the open-source publication of the tools we developed; and, (*iii*) the assessment of these heuristics using a proprietary extensive dataset of labeled addresses, which achieve precision levels of 1.0 and 0.979 respectively.

I. Introduction

In recent years, the rise of cryptocurrencies has revolutionized the global financial landscape, offering new opportunities for secure and decentralized transactions [1]. The first of these digital assets, Bitcoin [1] was quickly adopted by criminals due to its pseudonymous nature [2]. Law enforcement agencies are facing new challenges to detect illicit activities such as money laundering, terrorism financing, drug trafficking, and cybercrime [3]. Although there has been a noticeable shift of criminals from Bitcoin to other cryptocurrencies, billions of illicit funds continue to flow through Bitcoin [4].

When a Bitcoin address (i.e. unique identifier for a sender or a recipient of funds) is known to be associated with illicit activities, two common questions arise: Who is the entity responsible for the address, and which other addresses are under the control of this entity? Clustering heuristics delve into the public blockchain data to identify transaction patterns, address characteristics, unique transaction behaviors, and other elements that might inadvertently reveal information about the true owner of these addresses. By grouping together related addresses that are likely controlled by the same entity, clustering heuristics have emerged as a powerful tool for law enforcement agencies in their pursuit of combating financial crimes.

Numerous heuristics have already been established [5], [6], [7], [8], [9], [10], [11], [12], [13]. Regrettably, it seems that for every heuristic developed, a set of counter-techniques exist [14], [15], [16], [17]. This dynamic underscores the necessity for ongoing efforts to innovate and create new strategies. This paper introduces the implementation and evaluation of two novel Bitcoin address clustering heuristics: the Non-Standard Scripts (NSS) heuristic and the Public Elliptic-curve Key re-Employment (PEKET) heuristic.

Bitcoin relies on a script language named SCRIPT to specify the conditions under which an expense can be made from an address [18]. NSS will expose how some services can be detected by analyzing the scripts on Bitcoin. Apart from SCRIPT, Bitcoin makes use of asymmetric cryptography. Despite the recommendation to utilize a public key only once [1], this practice is far from being widely implemented. This paper demonstrates how services that reuse their public keys can be targeted by PEKET, which can unveil the service addresses.

This paper also evaluates the performance of both NSS and PEKET by relying on a ground truth dataset made of a set of addresses labeled with their true owners. Both our heuristics scope services' activity with great precision. Specifically, NSS achieves a precision of 1.0, while PEKET's precision can reach a maximum of 0.979. Moreover, the tools we developed to parse the scripts and extract the public keys will be released upon paper acceptance.

This paper is structured as follows: Sec. II introduces the fundamentals needed to understand the remainder of this paper; Sec. III positions our work with respect to the state of the art in Bitcoin clustering; Sec. IV introduces the two new clustering heuristics; Sec. V-A discusses our ground truth and the accuracy metrics used to assess our heuristics; Sec. V-B evaluates them; finally, Sec. VI concludes this paper by summarizing its main achievements.

II. Background

This section provides the required background for the remainder of this paper. In particular, Sec. II-A clarifies the term *Bitcoin* (peer-to-peer network vs. protocol vs. cryptocurrency). Sec. II-B focuses on the language implemented in the Bitcoin protocol to verify the transactions' validity.

A. The Bitcoin Protocol

The *Bitcoin Protocol* (BTC) implements a public, decentralized, and immutable ledger of financial transactions [1]. Contrary to the conventional banking system, its decentralized nature permits people to transfer funds without the involvement of a trusted third party entity. Money is represented by a virtual currency called *bitcoin* (₿), which equals one hundred million *satoshis* (the smallest subdivision of a ₿).

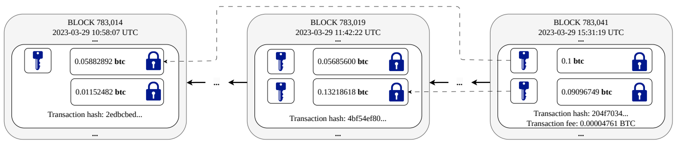

Fig. 1: The satoshis unlocked by the two inputs in the right-most transaction are split between two outputs. The transaction fee is attributed to the miner who includes the transaction into the block.

The Bitcoin network is a peer-to-peer network of nodes running the same consensus algorithm to relay transactions and update the *ledger*. The ledger is made of blocks, each of which contains up to 4MB of transactions [19]. These blocks are chained together to form the *blockchain*. The process of creating new blocks is called *mining*. The nodes participating in that process are thus called the *miners*. The BTC protocol is tuned to produce a block every 10 minutes, on average [20, Chapter 10]. Every miner has a probability of being chosen to produce the next block containing the latest transactions broadcast on the network that is proportional to the computing resources they allocate [20, Chapter 10].

In order to incentivize people to participate and to remove the presence of a central authority responsible for printing money, a given number of ₿ is created and rewarded to the nodes creating new blocks. The initial reward was set at 50 ₿ and is halved every 210,000 blocks [20, Chapter 10].

A transaction is made up of one set of inputs, one set of outputs, an optional data structure called the *witness* [19], and a few metadata fields, e.g., the version, the flags, etc [21]. An example of three transactions in three distinct blocks is provided in Fig. 1. An output represents some unspent money. It is defined by a value in satoshis and a script that defines the conditions for the money to be spent. On the other hand, an input refers to an output in a previous transaction and carries the data that is supposed to fulfill the conditions. When a new transaction is published and broadcast over the network, the miners are in charge of verifying, for every input, that the conditions are satisfied and that it does not point to an already spent output. In our example, the right-most transaction spends two outputs, respectively worth 0.05685600 ₿ and 0.13218618 ₿. This money is split over two new outputs, containing 0.1 ₿ and 0.09096749 ₿. The miners also ensure that the sum of the input values is greater than or equal to the sum of the output values. The difference between the two is the *transaction fee*, which is attributed to the node that has mined the block.

B. *The Bitcoin Script Language*

BTC supports its own programming language called SCRIPT. While it is not possible to write loops in this language [18], it still supports a range of arithmetic, cryptographic, and hash functions. Code written in SCRIPT can also contain conditional branches thanks to the operators: OP_IF, OP_NOTIF, OP_ENDIF, OP_ELSE, OP_RETURN, etc.

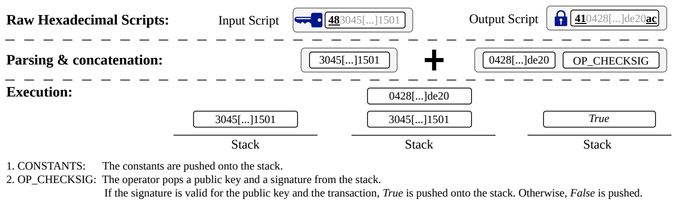
Raw Hexadecimal Scripts:
Input Script
483045[...]1501
Output Script
410428[...]de20ac
Parsing & concatenation:
3045[...]1501
+
0428[...]de20
OP_CHECKSIG
Execution:
3045[...]1501
Stack
0428[...]de20
3045[...]1501
Stack
True
Stack
1. CONSTANTS: The constants are pushed onto the stack.
2. OP_CHECKSIG: The operator pops a public key and a signature from the stack.
If the signature is valid for the public key and the transaction, True is pushed onto the stack. Otherwise, False is pushed.

Fig. 2: Claim of a PUBKEY output.

This stack-based language is employed to define the conditions under which an input is allowed to spend an output. Every output and input contain a script. These two scripts are concatenated and executed. The input is allowed to redeem the output if the execution terminates without any errors and returns `True`. Again, the miners are in charge of running the code and verifying the validity of the inputs.

A very simple example is provided in Fig. 2. Starting with the input script, the first hex byte (**48** – 72 in decimal) indicates the presence of a 72-byte constant. The decoding of the output script remains simple, the first hex byte (**41** – 65 in decimal) indicates the presence of a 65-byte constant. The second hex byte (**ac**) represents the operator OP_CHECKSIG.

The execution is straightforward: the two constants are pushed onto the stack, then the operator OP_CHECKSIG is executed. The two constants are popped from the stack. The top constant, the one that was in the output script, is a public key. Then, the operator checks that the constant that was in the input script is a valid digital signature with respect to the current transaction and public key.

Programming skills are not required to use BTC as a set of secured script patterns are defined: PUBKEY (PK), PUBKEYHASH (PKH), MULTISIG (MS), SCRIPTHASH (SH), WITNESS V0 KEYHASH (WKH), WITNESS V0 SCRIPTHASH (WSH), WITNESS V1 TAPROOT (WTR) [22], [23], [19]. These scripts are said to be standard.

As these standard scripts do not cover exhaustively all the potential use cases, some users prefer to conceive their own scripts to lock their money [24], despite the associated risks [25].

While Fig. 2 shows that a PUBKEY script defines one owner and requires its signature, SCRIPTHASH and WITNESS V0 SCRIPTHASH scripts are different in the sense that they do not define the conditions for the output to be unlocked. Instead, a second script, called the *redeem script*, is involved in the input side and defines the requirements for the output to be spent. This redeem script can be either standard or non-standard.

C. *Bitcoin Address*

Finally, it is essential to provide the formal definition of a BTC *address*.

A BTC address is the textual representation of all the information required to create an output locked with a standard script. Consequently, non-standard outputs and inputs do not have an address. It is worth mentioning that both SCRIPTHASH and WITNESS V0 SCRIPTHASH have an address even if their redeem script is non-standard. Thanks to BTC addresses, Alice can communicate to Bob how he should build the output to send her money. Thus, the address denotes all the outputs protected with this script and the inputs spending them.

III. Related Work

This section aims at discussing the clustering heuristics state of the art according to a taxonomy. The *transaction-scoped* heuristics (in Sec. III-A) comprise heuristics designed to cluster outputs or inputs that pertain to a single transaction. Conversely, *agnostic* heuristics (in Sec. III-B) encompass heuristics that do not necessarily take into account a single transaction, including NSS and PEKET. Lastly, in Sec. III-C, we outline prior research relevant to this manuscript, although these studies do not present a heuristic approach.

A. Transaction Scoped Heuristics

1) The Co-Spending Clustering Heuristic: Because BTC users spend some of their outputs to buy services or goods and collect the change in new outputs, the average value of their outputs necessarily diminishes. They eventually need to use several inputs within the same transaction to collect all the small change outputs.

This has been the foundation stone of the co-spending heuristic, which is probably the most frequently cited heuristic in the literature [5], [6], [7], [8](a.k.a. common-input-ownership, a.k.a. multi-input heuristic, a.k.a. cospend heuristic). As a rule of thumb, the inputs' addresses within the same transaction can be assumed to be controlled by the same entity. Satoshi himself mentioned this privacy issue [1].

To mitigate this leak of information, there exist obfuscation techniques such as *coinjoin transactions* or mixers.

A coinjoin transaction is a transaction in which the inputs are controlled by distinct entities [20, Chapter 6]. Some services simplify and automate the process for entities to pool their funds together into one transaction, thereby creating a coinjoin transaction [15], [16], [17]. Detecting those transactions is crucial because ingesting them would cluster addresses controlled by distinct entities.

Mixing services, on the other hand, act as intermediaries to make it difficult for observers to establish direct links between sender and receiver. They collect coins from users and send them coins that were deposited by other users [26, Chapter 6].

Failing to detect mixing and coinjoin transactions leads to the production of *superclusters* [9], i.e., clusters with a large number of addresses belonging to distinct entities. Thus, to be effective, the co-spending heuristic requires a lot of upstream work in order to analyze and filter the transactions being used by the heuristic.

2) The Change Output Clustering Heuristics: While the co-spending heuristic aims at grouping input addresses, the change output heuristics try to identify which output addresses correspond to the change outputs.

The one-time change heuristic supports that in transactions with two outputs o_n and o_e, with o_n being a new address (i.e., not linked to any past activity) and o_e being an address used in the past, the new address o_n is likely to be the change output address [5]. Being known to produce false positives [7], this heuristic has been refined multiple times [6], [7].

Many other heuristics have been proposed in the literature. The optimal change heuristic [10] claims that if a transaction has a unique output with a smaller value than any of the inputs, it is very likely to be the true change output. The round number heuristic [11] assumes that payment amounts are round numbers and that the leftover change amount would then be a non-round number. Finally, wallet fingerprinting heuristics [11], [12], which aim at identifying which wallet software was used to produce a transaction, can be used to detect change outputs because a change output is the one spent with the same wallet fingerprint.

Kalodner et al. [12] provide evidence of the privacy-infringing side-effect of multisig scripts use. In transactions where exactly one output matches the type of all input addresses, they tag this output as the change output. They evaluate the scope, i.e., the number of transactions and ₿ volume, of this change output heuristic but do not provide any metrics about the clusters that would be produced. Similarly, they implement deduplication based on underlying keys, they but did not quantify the impact of this approach.

B. Agnostic Heuristics

Strehle and Steinmetz [13] exploit the information in NULLDATA (ND) outputs, i.e., outputs which are provably unspendable and whose only purpose is to store data, to detect services' activity. They were able to link almost 32.4M transactions published between September 14, 2018, and December 31, 2019 to 37 distinct blockchain services.

C. Other

Bistarelli et al. [24] parse the blockchain up to block number 550,000 and analyze the non-standard scripts. They report statistics about their use and discuss their semantics. However, there is no proposal to exploit this form of intelligence to cluster addresses.

IV. NSS & PEKET Heuristics

We have seen in Sec. III that many heuristics already exist and cover a substantial part of the blockchain. However, they tend to generate false positives and negatives as effective obfuscation techniques exist. In the case of the co-spending heuristic, many techniques need to be implemented in order to detect coinjoin or mixing transactions [15], [16], [17].

This paper introduces two innovative clustering heuristics with very high precision for BTC. The first one, NSS, targets the use of non-standard scripts (Sec. IV-A). Their limited utilization allows for the accurate detection of services that operate on top of BTC. PEKET is more broadly applicable to the transaction history (Sec. IV-B). We will show how the reuse of the same public key can degrade privacy. Both of them belong to the category of agnostic heuristics defined in Sec. III.

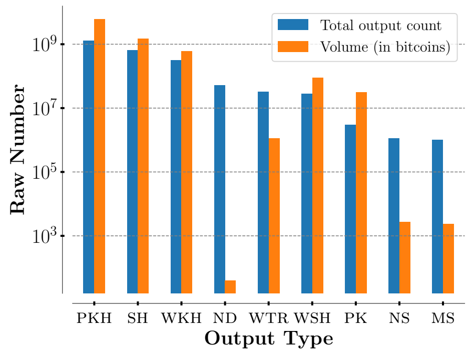

Fig. 3: Output count and total value locked per type. All acronyms are synthesized in the Appendix A.

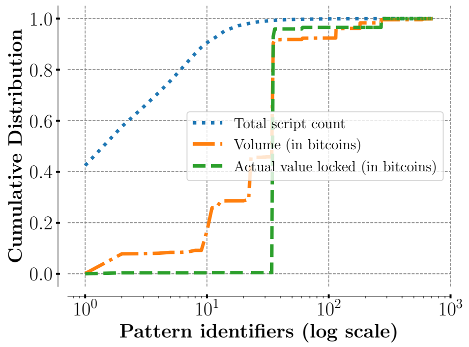

Fig. 4: CDF of multiple metrics for the script patterns ordered by frequency of use: $x = 1$ being the most frequently used pattern.

A. *Non-Standard Scripts Heuristic:* NSS

As a reminder from Sec. II, non-standard (NS) scripts can be found in two places. Firstly, in non-standard outputs, and secondly, as redeem scripts in SCRIPTHASH and WITNESS V0 SCRIPTHASH inputs. Fig. 3 illustrates the distribution of the output types over two dimensions (see Sec. V-A for our dataset description): the number of occurrences and the total value that has been locked. In total, 3.9M instances of distinct non-standard scripts have been found in the blockchain. These scripts were extracted from 1.1M non-standard outputs and 4.4M SCRIPTHASH and WITNESS V0 SCRIPTHASH inputs which are using non-standard redeem scripts.

In comparison, there are 2.4B outputs in BTC. This clearly shows that non-standard scripts are not widely spread and triggers questions about their usage, such as the possibility of linking their use to the activity of some entities.

In fact, some services are known to use non-standard scripts. For example, the company CHECKSIG provides a list of transactions as proof of reserve [27]. We have randomly selected and analyzed five of their WITNESS V0 SCRIPTHASH addresses:

- bc1qp5w9z6v6ljw00wqn[...]3y0v;
- bc1qak8fvqzkzqq0wg4a[...]mhmv;
- bc1q0wk2apulgmd982ss[...]vyjn;
- bc1qqst9un5sz8576fy2[...]resu;
- bc1qmqsa9cfxntlslru8[...]7snh.

All their redeem scripts share the same structure and opcodes. The difference resides in the constants being employed. We can define a *script pattern* as the script where all the constants are replaced with the same placeholder. Their common non-standard pattern is illustrated in Fig. 5.

This pattern is composed of two branches, with each part outlining a distinct method to access the funds. The first one includes six public keys. To unlock the money, three signatures are required. The second part includes three public keys and defines a second option for unlocking the money. It can be unlocked with two signatures if enough time separates the output and the input [18].

From the 3.9M distinct non-standard scripts, less than 800 different patterns are being used. Some patterns have been used

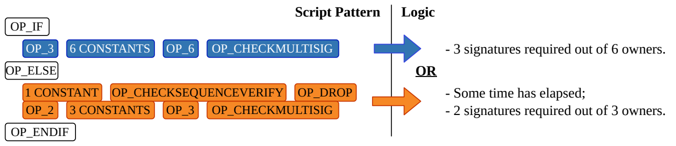

Fig. 5: Non-standard script pattern used by CHECKSIG.

once; the most frequently used pattern was used by 1.65M scripts. Fig. 4 shows the CDF of patterns. We observe that the first patterns are the most frequent ones. Additionally, some of them historically locked up a large amount of ₿ (also shown on Fig. 4). The second most frequent pattern used to secure a bit less than 200k ₿. From this volume, 384 ₿ are still currently protected with this pattern.

The volume and number of scripts could indicate that these patterns are or were being used by services.

The first heuristic we propose, NSS, is to cluster all the addresses that are protected with the same pattern. The heuristic is described by Algorithm 1. Since non-standard outputs and inputs lack an address, we assign them one in a manner that ensures multiple non-standard outputs that are locked with the same script would share the same address.

B. *Public Elliptic-curve Key re-Employment Heuristic:* PEKET

Satoshi recommends using a different key pair for each transaction to keep them from being linked to a common owner [1].

However, this advice is far from being applied by everyone. For example, considering the activity of this address 1HckjUpRGcrrRAtFaaCAUaGjsPx9oYmLaZ, the owner's public key, *030651e1[...]359d*, has been used millions of times. In total, 9.4% of all BTC addresses have been used more than once and represent more than 54% of all BTC outputs (see Fig. 6).

In the remainder of this section, we will demonstrate how the activity of services can be exposed through the reuse of their public keys.

1) Shared ownership outputs: BTC also supports outputs whose control is shared among N participants by using MULTISIG scripts [22]. Similarly to non-standard scripts,

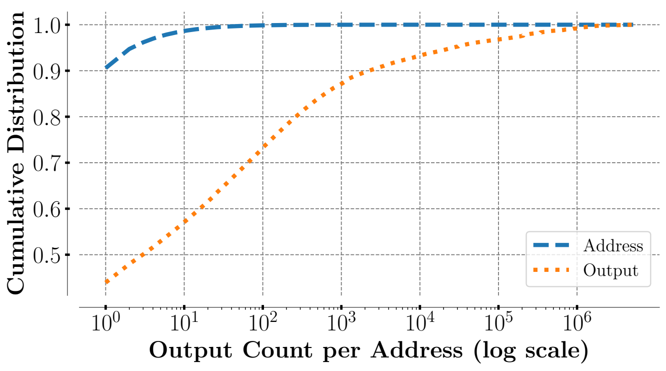

Fig. 6: Cumulative distribution of the addresses and outputs based on the frequency of use of the addresses. Frequency of use is expressed in terms of number of outputs per address.

MULTISIG can be found in two places: in MULTISIG outputs or in SCRIPTHASH and WITNESS V0 SCRIPTHASH inputs. We found that 9% of all BTC addresses involve multiple owners. The logic remains the same whether they are used as output scripts or redeem scripts.

This script defines N participants, and each participant includes its public key in the script. The script also includes an integer M with $M \leq N$. In order to spend the money, M out of the N participants need to include their signatures in the input.

This mechanism allows for several use cases. For example, escrow payments can be implemented by using a 2-3 MULTISIG ($N = 3, M = 2$). Two participants, let us call them Alice and Bob, and a third party, Carole, include their keys in the script. As the script requires two signatures, no participant can spend the money without the authorization of the others. In case of a dispute, Carole can sign a transaction with Alice to send the money to an address she fully controls [28].

These scripts can also be used by a single entity to increase security. The Bitmex company is known to use 3-4 MULTISIG redeem scripts in SCRIPTHASH addresses [29]. All their MULTISIG redeem scripts include the same three public keys. The fourth key is always different. Three signatures out of these four keys are required. The reuse of the same three public keys allows for the detection of all the addresses (thus all the outputs) the company handles, although Bitmex's concern is not privacy.

In clustering, we generally do not care about the individuals, and we focus on detecting services. For example, if a darknet market DM is using 2-3 MULTISIG to provide escrow payments between sellers and buyers, it is generally accepted to cluster all the addresses in the same cluster and tag them as managed by the darknet market.

2) Sole ownership outputs: The standard scripts PUBKEY, PUBKEYHASH, and WITNESS V0 KEYHASH define one owner by including their public key in the script. They can also be used as redeem scripts. In that case, the SCRIPTHASH and WITNESS V0 SCRIPTHASH are also controlled by a single pair of keys. Finally, MULTISIG scripts with $M = 1$ and $N = 1$ also define one owner.

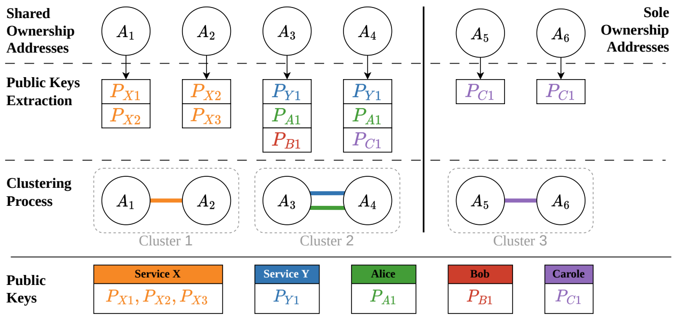

Fig. 7: An example of the clusters produced by PEKET.

It is evident that when multiple outputs share the same address, they also share the same owner(s) since the conditions to access the funds remain unchanged. However, the same public key can be included in different standard script types, e.g., the public key *0291a14d[...]f30b* is being used in a PUBKEYHASH and a WITNESS V0 KEYHASH output script. The logic of these two scripts is the same: it defines one owner and requires their signature to move the funds. But these two situations lead to two distinct addresses:

- 1MWhpubGujF7QUUzsdwey3gFrbz9qY7PRa
- bc1quyq23zmnnj8j2wpxyzpwyzr3ej6hugx5kdzf7z

As a result, these two addresses are owned by the same entity as they are entirely controlled by the identical pair of private and public keys. BTC addresses that involve a single owner represent 91% of all addresses.

3) The Heuristic: Algorithm 2 in the appendix expresses in pseudocode how to extract the public keys from any pair of input and output. We restrict the scope of this heuristic to standard scripts. Non-standard scripts are not widely adopted, as seen in Sec. IV-A, and the extraction of the public keys requires more work. It must be noted that we also exclude the public keys from WITNESS V1 TAPROOT outputs, which account for 1.4% of all outputs, because these keys result from the aggregation of the participants' public keys [30]. Thus, they do not reveal participants' identities.

The proposed heuristic is illustrated in Fig. 7.

- The addresses A_1 and A_2 should be clustered together as the public key P_{X2} is involved in both addresses;
- The addresses A_3 and A_4 should be clustered together as the public keys P_{Y1} and P_{A1} are involved in both addresses;
- The addresses A_5 and A_6 should be clustered together as Carole's public key P_{C1} is involved in both addresses.

However, clusters 2 and 3 should not be merged because cluster 3 groups Carole's addresses, while the second cluster groups the addresses resulting from the activity of service Y. Hence, sole ownership and shared ownership outputs should be treated separately.

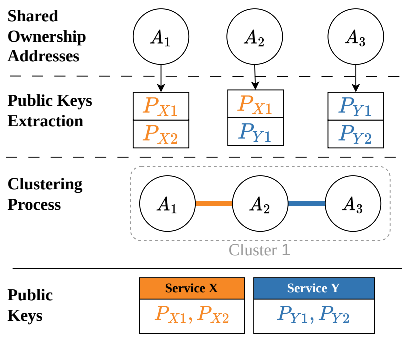

Fig. 8: A collision when 2 services share an address.

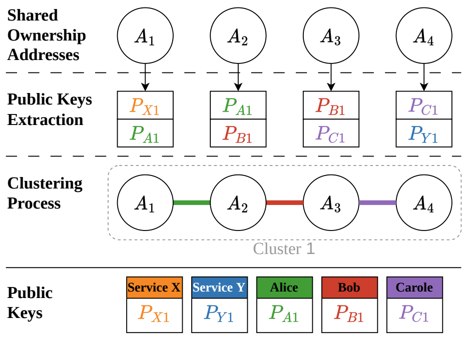

Fig. 9: A collision under the existence of a chain of user shared addresses.

While the clustering of the sole ownership addresses is safe, the clustering shared addresses can still create collisions. We are able to forecast two scenarios.

- The first scenario, depicted in Fig. 8, represents a situation in which two services share at least one address. For instance, if Bitmex were to utilize one of the public keys typically designated for securing their proof of reserves with a darknet market service, all the proof of reserve addresses and the darknet market addresses would be grouped together in the same cluster.
- The other plausible scenario involves individuals and potential chains of shared addresses and is presented in Fig. 9. Suppose we have the following situation. Alice's and Carole's public keys are respectively used in two distinct services X and Y. Moreover, Bob shares an address A_2 with Alice and A_3 with Carole. Thus, the services' addresses A_1 and A_4 would end up in the same cluster. Fortunately, it would be easy to mitigate these collisions caused by individuals by discarding their public keys. One can distinguish services' and individuals' public keys based on the flat number of outputs in which the public key is involved. Indeed, a public key being used thousands of times requires some automation. As creating outputs requires money, it is likely that these outputs are part of a service that generates income. For it to be considered a service public key, it has to be involved in **N** distinct addresses. This threshold **N** can easily be tuned to reach the desired precision. The downside of this approach is the inability to detect services that frequently change their set of keys.

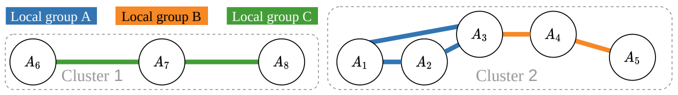

Fig. 10: An example of the clusters produced by a distributed connected component algorithm on a graph whose vertices are BTC addresses.

To sum up, the clustering heuristic can be formulated as:

1) For every public key pk:
   a) If pk is involved in more than 1 distinct sole ownership address,
      - Then, group all the sole ownership addresses controlled with pk.
      - Otherwise: no action required.
   b) If pk is involved in more than **N** distinct shared addresses,
      - Then, group the shared addresses.
      - Otherwise: no action required.

V. Evaluation

A. Ground Truth and Method

This paper relies on the BTC data and infrastructure provided by TRM Labs. They provided us access to BTC history from block 0 to block 795,000. The data originates from full BTC nodes [31] that they run. They parse the raw data and ingest it into their data warehouse, which is hosted by a cloud service provider.

No additional information besides what the nodes provide is used during that process. Thus, all experiments made in this paper can be reproduced by anyone running a BTC node.

The construction of clusters is achieved through the utilization of a distributed connected components algorithm [32, Chapter 21]. The process is illustrated in Fig. 10. First, we build a graph for which the nodes are the BTC addresses. An edge is created between two nodes whenever there is a desire to group these two addresses together within a cluster, regardless of the specific reason. For example, we aim to cluster addresses A_1,A_2, and A_3 for reason A and addresses A_3, A_4, and A_5 for reason B. As a result, the final cluster comprising addresses A_1,A_2,A_3,A_4, and A_5 is generated.

Our experiments were conducted on a cluster of four machines, each equipped with 4 vCPUs and 15 GB of RAM. Less than one hour was necessary to produce the clusters for the heuristics described in Sec. IV.

Additionally, our industrial partner provided us access to a table of high-confidence labels. A label is the identifier of the entity that owns a specific BTC address. Different sources are used to obtain these labels, such as their investigations team or various web scraping tools that they developed. Thereafter, the validity of these labels is checked by cross-checking the sources of information, collaborating with big actors in the industry and the implementation of automatic processes to double-check them. These labels were used solely in order to evaluate the correctness of our heuristics and were not involved in the development and tuning of our heuristics.

A *collision* is defined as the presence of different entities within the same cluster. Considering a clustering heuristic H producing a set of N clusters $C_H = \{c_i | 0 \leq i \leq N\}$ with $|c_i|$ being the number of addresses in the cluster c_i, we can also define two subsets:

- $C_{labeled}$: a subset of C_H, defined as the set of clusters for which we have at least one label;
- $C_{collision}$: a subset of $C_{labeled}$, defined as the set of clusters in which there is a collision.

Consequently, *precision* expresses the ratio of addresses that are in clusters that do not have any collisions and is defined as:

$$\forall c_l \in C_{labeled}, \forall c_c \in C_{collision}:$$

$$precision = \frac{\sum |c_l| - \sum |c_c|}{\sum |c_l|} \Rightarrow precision \in [0,1]$$

Additionally, the ground truth coverage (*GTCov*) expresses the ratio of addresses that are in clusters for which we have at least one label and is defined as:

$$\forall c_l \in C_{labeled}, \forall c \in C_H:$$

$$GTCov = \frac{\sum |c_l|}{\sum |c|} \Rightarrow GTCov \in [0,1]$$

B. Results

With our ground truth, we are able to evaluate the precision of the two heuristics proposed in Sec. IV. We also ran the co-spending heuristic [7] in order to compare it with our heuristics. This version runs without any mixing or coinjoin detection mechanisms.

For NSS, no collisions were found over the seven hundred clusters. The precision of this heuristic based on the provided data is thus 1.0. However, the coverage of this heuristic is quite limited. It only clusters 3.9M addresses out of 1.1B BTC addresses and accounts for 0.03% of the total ₿ volume. Out of these 3.9M addresses, 57.4% are in clusters for which we have at least one label ($GTCov = 0.574$). The size of the NSS clusters (in Fig. 12) can be taken as an indicator of the services' activities. Conversely, the patterns that are not in wide use could be the result of services' bugs that have been fixed or could be used by individuals.

On the other hand, PEKET is able to cluster up to 18M addresses (1.6% of all the addresses), which account for 5.6% of the total ₿ volume as illustrated in Fig. 11. This clearly shows that public key reuse is not a limited phenomenon. The precision increases with the threshold **N**, but at the cost of the volume and address coverage, as seen in the top half of Fig. 11. The heuristic with $\mathbf{N} = 10^3$ managed to cluster 11M addresses in 91k clusters with a precision of 0.978 and a ground truth coverage of 0.482. Moreover, **N** has a limited impact on the

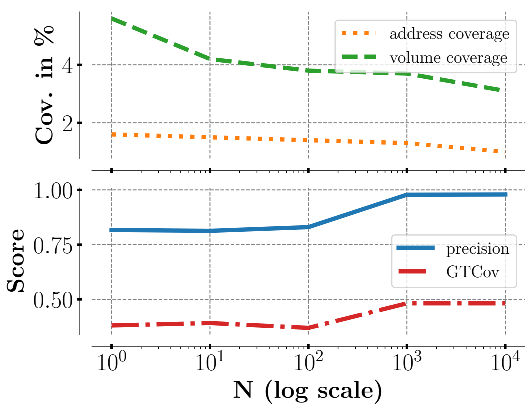

Fig. 11: Effect of **N** on the PEKET heuristic.

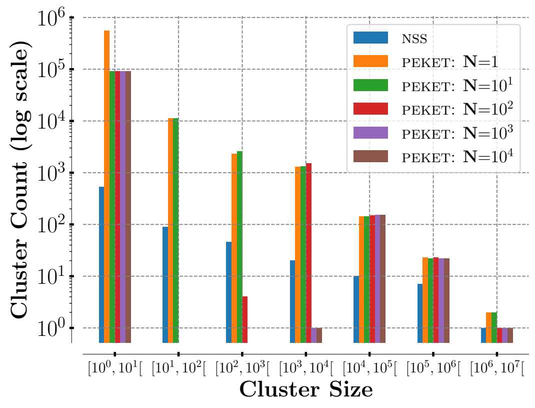

Fig. 12: Sizes of the clusters produced by PEKET and NSS.

clusters containing less than 10 addresses (see Fig. 12), as these clusters include mostly single ownership addresses.

These results have to be put in comparison with the co-spending heuristic, which reaches an address coverage of 63.5%, but at the detriment of a poor precision of 0.55. As discussed earlier, precision can be bumped up with the implementation of techniques to detect and discard coinjoin and mixing transactions, but at the cost of lower coverage.

Overall, these heuristics have proven to be reliable and can be easily combined with existing heuristics in order to detect services' activities.

VI. Conclusion

To conclude, the paper's two innovative Bitcoin address clustering heuristics have proven effective in detecting services with exceptional precision. Our research has made a contribution to the field of cryptocurrency investigations through the successful implementation and evaluation of these advanced techniques.

The importance of efficient Bitcoin address clustering cannot be underestimated, particularly in the context of combating financial crimes and detecting illicit activities within the cryptocurrency landscape. With the rising adoption of cryptocurrencies for both legitimate and criminal purposes, the need for robust investigative methodologies has become increasingly critical. Our heuristics offer a substantial step forward in this direction, enabling law enforcement agencies to detect services' activities and cluster their addresses.

Looking ahead, there is room for further enhancements, including extending the application of these techniques to other blockchains resembling BTC. Notably, certain blockchains such as DOGE [33], ZEC [34], etc, are based on forks from BTC source code. Additionally, exploring the extraction of public keys from non-standard scripts could expand the scope and capabilities of PEKET.

In closing, we firmly believe that the integration of these new Bitcoin address clustering heuristics into the toolkit of law enforcement agencies will enhance their capabilities of investigating and combating financial crimes.

ACKNOWLEDGMENTS

This work is supported by the CyberExcellence project funded by the Walloon Region, under number 2110186.

APPENDIX A
ABBREVIATIONS & ACRONYMS

The following list synthesizes all the abbreviations and acronyms defined and used in the paper.

- **PKH**: PUBKEYHASH.
- **SH**: SCRIPTHASH.
- **WKH**: WITNESS V0 KEYHASH.
- **ND**: NULLDATA.
- **WTR**: WITNESS V1 TAPROOT.
- **WSH**: WITNESS V0 SCRIPTHASH.
- **PK**: PUBKEY.
- **NS**: NON-STANDARD.
- **MS**: MULTISIG.

APPENDIX B
NSS HEURISTIC PSEUDOCODE

The following algorithm expresses in pseudocode the process used to group addresses that are secured with the same non-standard script pattern.

The main function CLUSTER_NSS groups addresses based on their pattern. The expected argument, *scripts*, contains all the outputs and redeem scripts extracted from the blockchain.

Three auxiliary functions are being used and their python implementation is available on `gitlab.uliege.be`.

Firstly, because the patterns of standard scripts are fixed and well-known, it is possible to determine the type of a script and implement a function CHECKTYPE, which takes as an argument a script and returns its type. If the script does not match any known standard pattern, its type is said to be non-standard.

Then, REPLACECONSTANTS takes a script as an argument, parses it, and replaces the constants with the same placeholder.

Finally, COMPUTEADDRESS derives an address from a script. For the redeem scripts, the process is already defined in the BTC protocol [19], [35]. Contrarily, non-standard outputs do not have an address, but it is possible to assign them a unique identifier based on the script by using a similar process.

Algorithm 1 NSS Heuristic

```
function CLUSTER_NSS($scripts$: list[array[bytes]])
    $clusters$ = dict<str, list[str]>()
    for $script$ in $scripts$ do
        $type = CheckType(script)$
        if $type$ == "NON STANDARD" then
            $pattern$ = ReplaceConstants($script$)
            $address$ = ComputeAddress($script$)
            $clusters[pattern]$.append($address$)
    return $clusters$
```

APPENDIX C
PUBLIC KEYS EXTRACTION ALGORITHM

The following algorithm expresses in pseudocode the process of extracting the public keys from any address. The data required to extract the public key varies from one type to another. This algorithm assumes access to one output and the input spending it. In particular, the expected inputs are:

- $scriptPubKey$: The output script.
- $scriptSig$: The input script.
- $witness$: The witness data in the input side.

Finally, our Python implementation of CHECKTYPE and EXTRACT is available on `gitlab.uliege.be`.

Algorithm 2 Public Keys Extraction

```
function EXTRACT($scriptPubKey$:array[bytes], $scriptSig$:array[bytes], $witness$:array[bytes])
    $type = CheckType(scriptPubKey)$
    if $type$ == "PUBKEY" then
        return $scriptPubKey.const[0]$
    else if $type$ == "PUBKEYHASH" then
        return $scriptSig[1]$
    else if $type$ == "MULTISIG" then
        return $scriptPubKey.const[1 : -1]$
    else if $type$ == "SCRIPTHASH" then
        if $witness.size() == 0$ then
            rdmScr = $scriptSig[-1]$
            return Extract($rdmScr, scriptSig[: -1]$, [])
        else if $scriptSig[1].size() == 20$ then
            return $witness[1]$
        else if $scriptSig[1].size() == 32$ then
            rdmScr = $witness[-1]$
            return Extract($rdmScr, witness[: -1]$, [])
    else if $type$ == "WITNESS V0 KEYHASH" then
        return $witness[1]$
    else if $type$ == "WITNESS V0 SCRIPTHASH" then
        rdmScr = $witness[-1]$
        return Extract($rdmScr, witness[: -1]$, [])
    return null
```

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," October 2008. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] Wikipedia Community, "Silk road (marketplace)," last Accessed: 29.08.2023. [Online]. Available: https://en.wikipedia.org/wiki/Silk_Road_(marketplace)

[3] Europol, "Cryptocurrencies: Tracing the evolution of criminal finances." last Accessed: 24.07.2023. [Online]. Available: https://www.europol.europa.eu/cms/sites/default/files/documents/Europol%20Spotlight%20-%20Cryptocurrencies%20-%20Tracing%20the%20evolution%20of%20criminal%20finances.pdf

[4] TRM Labs, "Trm labs' illicit crypto ecosystem report shows crime moving beyond bitcoin," last Accessed: 29.08.2023. [Online]. Available: https://www.trmlabs.com/report

[5] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *Proc. Financial Cryptography and Data Security (FC)*, April 2013.

[6] D. Ermilov, M. Panov, and Y. Yanovich, "Automatic bitcoin address clustering," in *Proc. IEEE International Conference on Machine Learning and Applications (ICMLA)*, December 2017.

[7] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: Characterizing payments among men with no names," *Communications of the ACM*, vol. 59, no. 4, pp. 86–93, March 2016.

[8] H. Xi, H. Ketai, L. Shenwen, Y. Jinglin, and M. Hongliang, "Bitcoin address clustering method based on multiple heuristic conditions," in *IET Blockchain*, vol. 2, June 2022, pp. 44–56.

[9] M. Möser and A. Narayanan, "Resurrecting address clustering in bitcoin," in *Proc. Financial Cryptography and Data Security (FC)*, October 2022.

[10] D. N. Jonas, "Data-driven de-anonymization in bitcoin," Master Thesis, ETH, August 2015.

[11] Bitcoin Community, "Privacy," last Accessed: 04.07.2023. [Online]. Available: https://en.bitcoin.it/Privacy#Change_address_detection

[12] H. Kalodner, M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator, and A. Narayanan, "BlockSci: Design and applications of a blockchain analysis platform," in *Proc. USENIX Security Symposium*, August 2020.

[13] E. Strehle and F. Steinmetz, "Dominating op returns: The impact of omni and veriblock on bitcoin," in *Journal of Grid Computing*, December 2020.

[14] Belcher, C., "Design for a coinswap implementation for massively improving bitcoin privacy and fungibility," last Accessed: 07.09.2023. [Online]. Available: https://gist.github.com/chris-belcher/9144bd57a91c194e332fb5ca371d0964

[15] Samourai Wallet, "Whirlpool," last Accessed: 20.07.2023. [Online]. Available: https://samouraiwallet.com/whirlpool

[16] zkSNACKs, "Wasabi wallet," last Accessed: 20.07.2023. [Online]. Available: https://wasabiwallet.io/

[17] Samourai Wallet, "Joinmarket-org/joinmarket-clientserver," last Accessed: 20.07.2023. [Online]. Available: https://github.com/JoinMarket-Org/joinmarket-clientserver

[18] Bitcoin Community, "Script," last Accessed: 27.06.2023. [Online]. Available: https://en.bitcoin.it/wiki/Script

[19] E. Lombrozo, J. Lau, and P. Wuille, "Segregated witness (consensus layer)," Bitcoin, BIP 141, December 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki

[20] A. Antonopoulos, *Mastering Bitcoin*. O'Reilly Media, Inc., 2014.

[21] Bitcoin Community, "Transaction," last Accessed: 27.06.2023. [Online]. Available: https://en.bitcoin.it/wiki/Transaction

[22] G. Andresen, "M-of-n standard transactions," Bitcoin, BIP 11, December 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki

[23] ——, "Pay to script hash," Bitcoin, BIP 16, January 2012. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki

[24] S. Bistarelli, I. Mercanti, and F. Santini, "An analysis of non-standard bitcoin transactions," in *Proc. Crypto Valley Conference on Blockchain Technology (CVCBT)*, June 2018.

[25] V. Jacquot and B. Donnet, "Chaussette: A symbolic verification of bitcoin scripts," in *Proc. International Workshop on Cryptocurrencies and Blockchain Technology (CBT)*, September 2023.

[26] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[27] Checksig Company, "Proof of reserves," last Accessed: 20.07.2023. [Online]. Available: https://www.checksig.com/por

[28] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," in *Financial Cryptography and Data Security*, April 2017, pp. 321–339.

[29] Bitmex Company, "Proof of reserves & liabilities," last Accessed: 20.07.2023. [Online]. Available: https://www.bitmex.com/app/porl

[30] P. Wuille, J. Nick, and A. Towns, "Base32 address format for native v0-16 witness outputs," Bitcoin, BIP 341, January 2020. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki

[31] Bitcoin Community, "Running a full node," last Accessed: 20.07.2023. [Online]. Available: https://bitcoin.org/en/full-node

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[33] Dogecoin Developers., "Dogecoin," last Accessed: 26.07.2023. [Online]. Available: https://github.com/dogecoin/dogecoin

[34] Zerocash Developers., "Zerocashd," last Accessed: 26.07.2023. [Online]. Available: https://github.com/Zerocash/Zerocashd

[35] P. Wuille and G. Maxwell, "Base32 address format for native v0-16 witness outputs," Bitcoin, BIP 173, March 2017. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki