Nonlinear computations on FinTracer tags

Michael Brand
School of Computing Technologies, RMIT University
Melbourne, Vic, Australia
AUSTRAC
Canberra, ACT, Australia
michael.brand@rmit.edu.au

Tania Churchill
College of Engineering & Computer Science, The Australian National University
Canberra, ACT, Australia
AUSTRAC
Canberra, ACT, Australia
tania.churchill@anu.edu.au

Carsten Friedrich
AUSTRAC
Canberra, ACT, Australia
carsten.friedrich@austrac.gov.au

ABSTRACT

Recently, the FinTracer algorithm was introduced as a versatile framework for detecting economic crime typologies in a privacy-preserving fashion. Under the hood, FinTracer stores its data in a structure known as the "FinTracer tag". One limitation of FinTracer tags, however, is that because their underlying cryptographic implementation relies on additive semi-homomorphic encryption, all the system's oblivious computations on tag data are linear in their input ciphertexts. This allows a FinTracer user to combine information from multiple tags in some ways, but not generically. In this paper, we describe an efficient method to perform general nonlinear computations on FinTracer tags, and show how this ability can be used to detect a wide range of complex crime typologies, as well as to extract many new types of information, while retaining all of FinTracer's original privacy guarantees.

KEYWORDS

anti money laundering, financial crime, graph analytics, homomorphic encryption, private graph analysis

1 INTRODUCTION

There is no doubt that detecting financial crimes such as money laundering and fraud is easiest when all information regarding a particular crime is pooled together. However, this information is often siloed among the different financial institutions, and cannot be shared due to privacy, commercial, and in some countries also legal considerations. This real-world problem presents an opportunity for privacy-preserving algorithms. Collaborative protocols, which jointly compute functions of interest without invasion of privacy, have, indeed, been previously proposed for this purpose. (See, e.g., [13].)

The recent introduction of *FinTracer* [5] changes this landscape by providing an algorithmic primitive that is both lightweight and highly customisable, which can be used as a powerful building-block in the construction of larger, more sophisticated algorithms.

FinTracer is an algorithm based on the idea that information can be stored in a data structure known as a *FinTracer tag*, over which computations can then be performed securely. A FinTracer tag stores a single Boolean for each bank account in a candidate set V, and it is this Boolean that is then manipulated in the computation. Due to the lightweight nature of the FinTracer algorithm, the set V can be as large as all accounts, nation wide, even in an economy the size of Australia's.[1]

We expand on the FinTracer tag data structure later on, but for the purpose of introducing it, it is enough to note that information in a tag is stored in the form of a partial mapping from V to a semi-homomorphically-encrypted value. For reasons of execution speed, the algorithm uses ElGamal encryption [7, 11] over the additive Curve25519 [3] twisted Edwards elliptic curve group [4], and curve points are stored in their extended projective form [9].

Here, the plaintext corresponding to the stored ciphertext is an integer modulo the elliptic curve size (approximately 2^{252}), where zero encodes a "False", nonzero encodes a "True", and any element of V that is not part of the partial mapping is taken to be mapped to a default value, this default value usually being an encrypted zero, which is to say "False". We refer to these plaintext numbers as the *tag values*.

FinTracer tags are stored in a distributed fashion, partitioned among financial institutions, so that each financial institution only holds the part of the mapping relating to its own accounts. None of the financial institutions holds the cryptographic key for decrypting tag values. This is generated and kept by the Financial Intelligence Unit (FIU) that performs financial crime investigation using FinTracer.

In [5] it was shown that such a design for tags allows the computation of many functions of real-word interest for intelligence analysts investigating financial crime.

This design was inherently limited, however, in that if all values private to a single party are stored in the beginning of the computation as encrypted tag values, then due to the semi-homomorphic nature of the chosen underlying encryption scheme the only operations that can be computed obliviously over these values are linear functions of the private inputs.

All operations discussed in [5] can be described as linear functions over the private inputs, whose results for each account are then mapped to either `False` or `True` based on whether the linear function's output is zero or nonzero, and it is this Boolean that is ultimately revealed to the FIU. We refer to operations that can be described in this way as *linear operations*.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–14
© YYYY Copyright held by the owner/author(s).
https://doi.org/XXXXXXX.XXXXXXX

[1]Australia's Gross Domestic Product (GDP) was \$1.423T as of 2020 [6], making it the world's 13th largest economy.

In this paper, we show how nonlinear operations can be computed over FinTracer tags, and demonstrate the use of such nonlinear operations in substantially expanding FinTracer's abilities to be used in detecting financial crime.

1.1 The privacy assumptions

Although adaptable to other paradigms, the FinTracer algorithm was initially developed for the Australian financial market, and because of this its original description reflected the privacy constraints of that environment.

We will continue this, and describe the computation of nonlinear operations within the same constraint environment. Chiefly, these constraints can be described as follows.

The *Australian Transaction Reports and Analysis Centre* (AUSTRAC) is Australia's anti-money laundering and counter-terrorism financing regulator and financial intelligence unit. It is the government agency "responsible for preventing, detecting and responding to criminal abuse of the financial system to protect the community from serious and organised crime" [2].

The Anti-Money Laundering and Counter-Terrorism Financing Act 2006 (AML/CTF Act) [8] requires reporting entities (those entities that provide designated services—services which, under the Act, are identified as posing a risk for money laundering and terrorism financing) to maintain an AML/CTF program. Each reporting entity is required to provide a report to AUSTRAC on International Funds Transfer Instructions, cash deposits and withdrawals equal to or greater than AU\$10,000, and cross-border movements, as well as to submit Suspicious Matter Reports (SMRs).

Reporting entities are required to analyse their own data to identify any suspicious activity. If a suspicion is formed that an entity or transaction may be linked to a crime, an SMR must be submitted to AUSTRAC. Reporting entities do not have access to data from other financial institutions in their analysis process. In fact, they "must not disclose any information about an SMR, or do anything which could reasonably infer that [they] have submitted an SMR or are required to submit an SMR about one of [their] customers (except for certain limited circumstances)" [1]—to do so would in most circumstances be a criminal offence.

Consider, now, an algorithm working in an environment that contains AUSTRAC and some set F of financial institutions which are reporting entities to AUSTRAC. In view of the AML/CTF Act, an algorithm working in such an environment must face strict restrictions against any suspicions (or any grounds for suspicions) being leaked from one $f \in F$ to any other, either directly or indirectly, but AUSTRAC itself is empowered to receive information on suspicion, whether it can be formed by a single $f \in F$ or only by multiple financial institutions working in tandem.

Although AUSTRAC does have information-gathering powers under the AML/CTF Act, we want to design our algorithms so as to maximally protect the privacy of account owners for whom no suspicion of criminal behaviour exists. This being the case, we designed our algorithms so that AUSTRAC only receives in the course of running them either general statistics or information about accounts that are actively suspicious, in the sense that they match sought patterns of criminal activity. We refer to such patterns as *criminal typologies*.

The requirement for financial institutions to maintain their own AML/CTF programs gives each financial institution a strong incentive to learn of any suspicion that any collaborative algorithm may report to AUSTRAC about that institution's own clients. AUSTRAC can legally share this information.

The idea of FinTracer is to provide a distributed computation environment, where AUSTRAC and a set of participating financial institutions F each operate their own computation node, where AUSTRAC can query the system for any type of behaviour pattern indicating criminal activity, similar to what can be done in standard graph query languages. The algorithm must then retrieve for AUSTRAC instances of such typologies, even though these instances are composed of account and transaction data that may not all be available to any single party: account information will be known only to the financial institution managing the account, and transaction information will be known only to the financial institutions participating in the transactions.

In accordance with the above, such an algorithm must work within the following constraints:

- The algorithm must not supply to the AUSTRAC node any information beyond the requested query result and general, privacy non-invasive statistics.
- The algorithm must supply to each participating financial institution the portion of the result related to their own accounts, but is not allowed to supply to the financial institutions any other information.

Though our solution was initially built for the Australian financial market, and therefore tailored to Australian law, the descriptions above should be taken merely as describing a deployment example. The system is a general-purpose system with applicability anywhere where the legal requirements meet the constraints set out above. The role taken up by AUSTRAC in our Australian example is in this more general context known as a *Financial Intelligence Unit* (FIU).

Throughout the rest of the paper, we will therefore use generic terms, instead of referring specifically to AUSTRAC and to its reporting entities. In our descriptions, we will consider a system running on a network where one node is the FIU node, to be operated by the relevant FIU, and the rest of the nodes will be financial institutions from whom the FIU receives information. Figure 1 depicts the architecture of such a system. It is this architecture that we implemented and used in our experiments.

1.2 FinTracer and FinTracer tags

Let G be a digraph, $G = \langle V, E \rangle$, where E is a set of directed edges (a, b) in which a and b are accounts and $(a, b) \in E$ reflects a particular relation of interest that is manifested in the domestic transaction data between a and b. Such a relation may take into account both any transactions from a to b and any transactions from b to a. The set V, in turn, is the set of accounts induced by E.

A FinTracer query is a graph query, asked by the FIU, that is run on G. However, because the FIU has, per assumption, no knowledge of any *a priori* non-suspicious accounts or transactions, it is not able to describe G directly. Instead, the FIU formulates the nature of the relation to be described by E in the form of a query, Q_G, which each financial institution can interpret separately. This is normally conveyed as a SQL query.

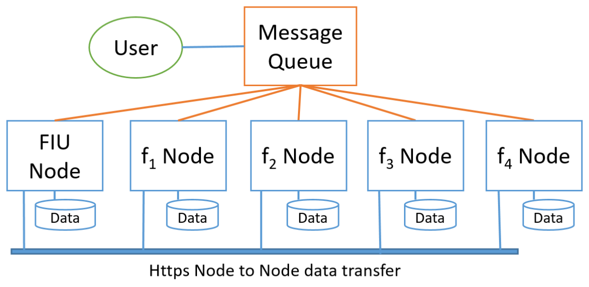

Figure 1: Architecture of the FinTracer system. One node is to be operated by the FIU, and each of the other nodes by one financial institution. The user querying the system is an FIU analyst.

Let $S, D \subseteq V$ be sets of accounts similarly described by queries Q_S and Q_D, respectively. We refer to S as the *source accounts* and D as the *destination accounts*.

The basic FinTracer query, which the FinTracer algorithm of [5] is able to resolve, is described by a tuple $\langle Q_G, Q_S, Q_D, k\rangle$, where k is a natural. The answer, A, is the subset of D whose distance on G from any $s \in S$ is at most k. We describe this relation as

$$A = \texttt{DistanceAtMost}(Q_G, Q_S, Q_D, k),$$

or algorithmically as

$$A \longleftarrow \texttt{DistanceAtMost}(Q_G, Q_S, Q_D, k).$$

The FinTracer algorithm allows one to answer such queries without impinging on the privacy of any non-matching account. We will use it as a building block to construct more advanced algorithms.

To simplify the presentation, throughout the remainder of this paper, where it is not ambiguous we will describe the inputs to FinTracer as the tuple $\langle G, S, D, k\rangle$, but it should be understood that this is shorthand for $\langle Q_G, Q_S, Q_D, k\rangle$: the values of G, S and D are generally not known to the FIU, so can only be defined by a description.

The answer delivered from the algorithm, A, first appears in the form of a FinTracer tag. As described above, a FinTracer tag is a data structure that stores a partial mapping from V to semi-homomorphically encrypted values, the plaintext of which indicates either zero for “False” or nonzero for “True”, with the default (for accounts not in the partial mapping) usually being “False”.

Importantly, FinTracer tags are stored in a distributed fashion: each financial institution only holds the portion of the tag related to its own accounts. Other financial institutions are not even aware of the existence of particular accounts, unless these accounts transacted with their accounts, and the FIU may not be aware of the existence of any specific account, unless a suspicion exists regarding it.

Additionally, the private key to decrypt any data in a tag is held exclusively by the FIU, so initially the existence of any such tag does not add any information to any party: all tag values are held by the participating financial institutions, but only the FIU holds the key to decrypt them.

In [5], it is shown how to compute the answer to a query as above in the form of a FinTracer tag (i.e., how to create, in a privacy-preserving manner, a tag such that exactly those $d \in D$ that match the query have nonzero values in the tag). It is also shown, separately, how the information from such a tag can be retrieved, in a way that ensures that the FIU receives only the information it is entitled to receive, and each financial institution receives exactly the information it should as well.

2 THE POWER OF NONLINEAR OPERATIONS

In the FinTracer algorithm, the value of the source accounts, S, can be provided in the form of a FinTracer tag. In such a case, whether any account appears in S may be information not directly available to any of the computing parties: no party other than the FIU possesses the private key with which the tag values are encrypted, and the FIU does not possess any part of the tag’s mapping.

The reason S can be given as a FinTracer tag is that FinTracer’s tag propagation mechanism relies exclusively on additive semi-homomorphic operations that are computed obliviously by the non-FIU nodes.

Such a mechanism is, however, intrinsically limited in that it is only able to compute linear operations. For FinTracer queries, specifically, this indicates that given fixed values for all other parameters of the FinTracer query (G, D and k), the relation between the tag values of S and the tag values ultimately returned to the FIU node must necessarily be a linear function.

Consider, however, operations that take multiple tags, each containing some Boolean information per account, and compute a new tag, where each account is associated with a new Boolean that is the result of applying a user-chosen Boolean function on the original tag values, where the new Boolean related to each account is computed only based on the Boolean values associated with the same account in the input tags.[2] Such *Boolean operations* may be linear or nonlinear, depending on the choice of Boolean function.

In this paper, we describe how to compute any Boolean operation, linear or nonlinear, in a privacy-preserving manner, over FinTracer tags.

In [5], it was shown how important criminal typologies can be formulated as basic FinTracer queries. However, these were necessarily typologies that could be described using only a single type of relationship that connects two accounts. On its own, FinTracer

(1) Cannot describe more complex typologies that include a larger number of relationships between a larger number of accounts, and
(2) Cannot retrieve for each found typology more information than the identity of the found set of accounts.

Before continuing to describe how Boolean tag operations can be performed under privacy preservation, we first show, in this section, some examples of the power of the combination of FinTracer propagation with Boolean operations, how this combination can be used to describe a diverse array of realistic criminal typologies that are not queryable using FinTracer alone, and how this combination allows users to answer many more questions that may be interesting in the context of financial crime investigation.

[2]Our description ignores the values associated with accounts that do not appear in the tag’s partial mapping. These values are, however, known to the financial institution by their very absence, so can be imputed where appropriate.

2.1 Chained operations

In its standalone formulation, FinTracer utilises two distinct data types: sets (such as S and D) for inputs, and tags (such as A) for outputs. Sets can be made into tags, and, in fact, the parameter S is only ever used as a tag so it can be communicated via a tag rather than via a set description.

The same is not true for D. In standalone FinTracer, the destination set, D, must be communicated as a set, known to the non-FIU computing nodes, because FinTracer works by initially computing a tag, T_k, of all accounts that can be reached from S using a walk on G of length at most k. FinTracer's information retrieval procedure then uses T_k and D to return to the FIU node the account values in T_k, but just over the domain D.

The Boolean intersection operation allows FinTracer to accept also its D parameter as a tag. In this case, the answer to be retrieved is simply $T_k \cap D$ (where Boolean operation notation indicates that this operation is applied account-by-account on both tags).

The ability to use a tag, potentially unknown to the non-FIU computing nodes, to define the destination set, D, creates a computing environment in which both FinTracer's inputs and outputs (excepting G and k) are tags. Sets, as a separate data type, are no longer needed in the system at all. In such an environment one can combine operations, chaining them together to create arbitrarily complex queries.

2.2 Describing complex typologies

Consider the following typology. We are seeking instances of triplets $\langle a, b, c\rangle$, such that a, b and c are accounts, where

- Cash was deposited into a and then funnelled into b, and
- The owner of b is a known associate of the owner of c who is a known person of interest.

Each of the above is a relationship that FinTracer can find, but the combination of the two is not. In order to find all b matching this description, one would need to follow a process such as the below:

- Find B_1, the set of all b_1 for which there exists an a such that the $\langle a, b_1\rangle$ pair holds the first sought relationship.
- Find B_2, the set of all b_2 for which there exists a c such that the $\langle b_2, c\rangle$ pair holds the second sought relationship.
- Retrieve $B = B_1 \cap B_2$ as the set of all b matching the typology.

Note that only the retrieval phase adds any information to the computing parties. The finding of the sets B_1 and B_2, as well as the finding of their intersection, is done obliviously.

The process depicted above is emblematic of how complex typologies can be described by stitching together simpler two-account typologies, but relies on the ability to perform the final intersection—this being a nonlinear operation.

For example, in the $\langle a, b, c\rangle$ example given above, once we have found B, we can use additional FinTracer runs that use the found B set as their source account set S in order to find all a and all c that participate in such a relationship. The full process is as follows.

- Find B_1, the set of all b_1 for which there exists an a such that the $\langle a, b_1\rangle$ pair holds the first sought relationship.
- Find B_2, the set of all b_2 for which there exists a c such that the $\langle b_2, c\rangle$ pair holds the second sought relationship.
- Find $B = B_1 \cap B_2$ as the set of all b matching the typology.
- Find A, the set of all a for which there exists a $b \in B$ such that the $\langle a, b\rangle$ pair holds the first sought relationship.
- Find C, the set of all c for which there exists a $b \in B$ such that the $\langle b, c\rangle$ pair holds the second sought relationship.
- Retrieve $\langle A, B, C\rangle$.

This is an example of a typology involving three *roles*, a, b and c, for each of which we can retrieve the full result set (A, B and C, respectively).

The same technique can be extended to retrieve the account sets of all roles in any typology that can be described by a tree of relationships.[3]

2.3 Exact Distance

A run `DistanceAtMost`(G, S, D, k) of the basic FinTracer algorithm only answers which subset of D is at distance at most k on G from any element in S.

An intelligence analyst investigating G can get a little more information by considering the inverted graph, G^{-1}, being the digraph over the same vertex set V, but with all edges running in the opposite direction to their direction in G. Running the FinTracer algorithm on $\langle G^{-1}, D, S, k\rangle$ answers the question of which subset of S connects to any element of D by a path of length at most k on G.

This, however, is basically all the information such an intelligence analyst can glean from the standalone FinTracer algorithm regarding the particular relationship described by $\langle G, S, D, k\rangle$.

Once nonlinear operations are allowed, however, more complex information can be retrieved.

To begin with (although this is perhaps not the example of highest practical value for financial crime investigation) consider how one may supplement the function "`DistanceAtMost`(G, S, D, k)" by a second function, "`DistanceExactly`(G, S, D, k)", that returns those accounts in D whose minimal distance on G from any account in S is *exactly* k.

In [5], a similarly-described function appears, but it is subtly different: it returns those accounts in D satisfying that there is a walk on G from some $s \in S$ to them that is of length k exactly. This is, however, not the same. `DistanceExactly` is, in fact, not a linear operation, so cannot be computed using the tools of [5] alone.

To see this, consider, for example, the graph $G = \langle V, E\rangle$ with $V = \{a, b, c\}$ and $E = \{(a, b), (b, c)\}$, which we will query with $D = \{c\}$ and $k = 2$. If we initialise S with nonzero $S(a)$ and $S(b)$ being zero (encrypted), the return result tag at c should be nonzero, but if we initialise $S(b)$ with any other value, the output should be zero. Such an operation cannot be linear.

Given Boolean operations, however, the question becomes trivial, as shown in Algorithm 1.

2.4 In-between accounts

Consider a relationship $\langle G, S, D, k\rangle$ of the kind describable with FinTracer. While `DistanceExactly` can discover the subsets $S' \subseteq S$ and $D' \subseteq D$ such that elements of S' are at distance exactly k on G

[3]Typologies involving cycles may have a result set that is not a Cartesian product of the account set results in each of their roles, so require a different approach. We discuss methods to tackle this issue in a separate upcoming paper.

Algorithm 1 Computing DistanceExactly(G, S, D, k)

```
1: if $k = 0$ then
2:     return DistanceAtMost$(G, S, D, k)$.
3: else
4:     $A \leftarrow$ DistanceAtMost$(G, S, D, k)$.
5:     $B \leftarrow$ DistanceAtMost$(G, S, D, k-1)$.
6:     return $A \setminus B$.                ▷ Where "\" is set difference.
7: end if
```

from some element in D' and vice-versa, in practice a financial intelligence analyst may want to retrieve the identities of all accounts on the paths connecting S' to D'.

With Boolean operations this is possible, and can be done using the following process.

(1) For all $j : 1 \le j < k$, find D^j, the set of all $v \in V$ whose distance from S on the digraph G is j.
(2) For all $j : 1 \le j < k$, find S^j, the set of all $v \in V$ whose distance from D on the digraph G^{-1} is j.
(3) Retrieve the union of all $D^j \cap S^{k-j}$ over $j : 1 \le j < k$.

3 THE ALGORITHM

We begin our description of how to compute Boolean operations from the simplest operations to compute, and work our way to general operations.

3.1 Union

One can equate the information contained in a FinTracer tag with a set of accounts, a subset of V, by considering each tag as representing the set of accounts that the tag maps to "True". This being the case, one can use set notation to describe operations on tags.

The simplest Boolean operation to compute on FinTracer tags is union. Given two tags, T_1 and T_2, we seek to compute $T = T_1 \cup T_2$, this being a tag mapping any account a to "True" if either T_1 or T_2 maps it to "True".

The reason this operation is simple to compute is that it can usually be translated to a linear operation: by simply summing the semi-homomorphically encrypted values in T_1 and T_2 corresponding to each account, a, the result can be used simply as the tag value for a in T. We denote such an operation by "$T \leftarrow T_1 + T_2$". This is an operation that can be performed by the individual financial institutions in an embarrassingly-parallel manner, without any need for communication.

The reason addition can be used as a way to compute union is that in FinTracer, a tag value of zero encodes "False" and nonzero encodes "True". If the underlying tags were nonnegative integers, addition would have corresponded directly with set union.

In reality, however, tag values are computed modulo the elliptic curve size, leaving the possibility for two nonzero values to sum together to exactly zero.

Where summed values are small, this is not a concern (given that the group size is on the order of 2^{252}), and even when not, the chance of such an event occurring accidentally is minute. Nevertheless, in some cases, e.g. if the operands come from a third party, we do want to protect the algorithm against the eventuality that the operands may have been chosen maliciously so as to sum to zero.

In this case, the solution is to *sanitise* one of the operands prior to the summation. Sanitisation is the operation of multiplying each tag value by an independent, uniformly-chosen, random nonzero integer modulo the elliptic curve group size. After sanitisation, each nonzero value is transformed to a uniformly-distributed nonzero value on the elliptic curve, meaning that the result of any summation with a nonzero value performed on it, even if maliciously chosen, has only approximately 2^{-252} probability of yielding a zero, this probability being entirely negligible for our purposes.

Such sanitisation also requires no communication, and can be performed independently at each financial institution.

3.2 Negation

The first nonlinear operation we tackle is negation. Negation is rarely useful on its own, but is often a key ingredient in more complex operations, such as when computing set difference.

While negation is not normally considered a nonlinear operation, in our context it is, because we consider operations on encrypted elliptic curve elements, rather than on Boolean values: the result of "$\neg T$" on a tag T will map any nonzero to a zero and any zero to a 1.

This is done as described in Algorithm 2. Here and throughout, we use "**on**" blocks to indicate which nodes code runs on. Lines of code that run on multiple nodes can run in parallel. The command "**transmit**" is used to indicate inter-node communication. It sends information from the executing nodes to designated target nodes. The command "**receive**" indicates receipt of the information at the target nodes. It is assumed that all such communication is cryptographically and/or physically protected at the channel level, to ensure that it cannot be intercepted other than by the sending and the receiving parties. All other commands run in embarrassingly-parallel fashion. We use "FIU" to indicate the FIU node, and F as the set of participating financial institutions.

Throughout, we take K_{pub} to be the public key that was used to encrypt tag values, which is assumed to be known to all parties, and K_{priv} to be the corresponding private key, known only to the FIU.

The operation "sanitise" was discussed in Section 3.1. The operation "refresh" is the addition of a freshly-encrypted zero to a ciphertext. See Appendix A for a discussion of the mathematics of refreshing and sanitisation, and [5] for implementation optimisations. Both are linear operations that do not require communication.

For a tag, T, we use T^f to indicate the portion of T that is visible to financial institution f. This is a tag representing the subset of T whose accounts are managed by f.

Note that the specific semi-homomorphic encryption system used here cannot be fully decrypted. Decryption yields an elliptic curve group element, not an integer. Line 18 of Algorithm 2 is performed by comparing the decrypted value with the zero in the elliptic curve group.

Furthermore, note that in the resulting tag, $\neg T$, the default value assigned to an account not in the mapping may be a "True", represented, e.g., as an encrypted "1", rather than as the usual "False" ("0"): the default value for $\neg T$ should be the negation of the default value of T.

THEOREM 1. *Algorithm 2 is correct.*

Algorithm 2 Computing $\neg T$

```
1:  on all $f \in F$ do:
2:      Sanitise $T^f$.
3:      Refresh $T^f$.                                  ▷ See Appendix A.
4:      Add noise to $T^f$.                             ▷ See Section 4.
5:      Let $\pi^f$ be a uniformly-chosen random bijection from
        $1, \ldots, |\mathrm{dom}(T^f)|$ to the domain of $T^f$.
6:      Let $v^f$ be a vector of encrypted elements of length
        $|\mathrm{dom}(T^f)|$.
7:      for all $i \in 1, \ldots, |\mathrm{dom}(T^f)|$ do
8:          $v^f[i] \leftarrow T^f(\pi^f(i))$.
9:      end for
10:     transmit $v^f$ to FIU.
11: end on
12: on FIU do:
13:     $zero \leftarrow Enc_{k_{\mathrm{pub}}}(0)$.          ▷ Store an encrypted "zero".
14:     $one \leftarrow Enc_{k_{\mathrm{pub}}}(1)$.           ▷ Store an encrypted "one".
15:     for all $f \in F$ do
16:         receive $v^f$ from $f$.
17:         for all $i \in 1, \ldots, |v^f|$ do
18:             if $Dec_{K_{\mathrm{priv}}}(v^f[i]) = 0$ then
19:                 $v^f[i] \leftarrow one$.
20:             else
21:                 $v^f[i] \leftarrow zero$.
22:             end if
23:         end for
24:         Refresh $v^f$.
25:         transmit $v^f$ to $f$.
26:     end for
27: end on
28: Let $R$ be a new empty tag.
29: on all $f \in F$ do:
30:     receive $v^f$ from FIU.
31:     for all $i \in 1, \ldots, |v^f|$ do
32:         if $\pi^f(i) \in V$ then                    ▷ See Section 4.
33:             $R^f[\pi^f(i)] \leftarrow v^f[i]$.
34:         end if
35:     end for
36: end on
37:                                 ▷ $R$ is the final result, equal to $\neg T$.
```

Proof. In Algorithm 2, the financial institutions send to the FIU the original tag values. These tag values are then negated by the FIU and placed back in their appropriate tag positions by the financial institutions. Hence, correctness of the algorithm is straightforward. □

Theorem 2. *Algorithm 2 does not reveal any new information to the financial institutions. It only reveals to the FIU information regarding the number of zero-valued tags and the number of nonzero-valued tags in the portion of tag T managed by each financial institution.*

Proof. The financial institutions only receive a vector of freshly-encrypted items of a length that they themselves determine. Hence, no information is passed to any f. The FIU receives tag values that have been refreshed, sanitised, combined with noise and permuted. Refreshing and sanitisation, as explained in Appendix A, ensure that the encrypted values themselves contain no information other than the Boolean value they are meant to convey. By permuting them using the random permutation π^f, their order becomes equally uninformative, so the information given to the FIU is only the total number of "False" and the total number of "True" values in the tag. □

The process of adding noise on Line 4 of Algorithm 2 ensures that no private information about any specific account is divulged to the FIU from these two sums. The specifics of such noise addition and proofs of the process's privacy guarantees are given in Section 4. Once such private information is purged, the remaining information, ultimately transmitted to the FIU, is only the approximate size of the number of "False" and "True" tag values. These two numbers qualify as "general statistics", which we assumed can be shared with the FIU.

We note that if several tags need to be negated, this can be done simultaneously, by use of a single long vector for their values and a single large permutation to scramble all encrypted values jointly. The noise-adding technique detailed in Section 4 will, in this case, require for the joint vector less noise than would have been required to negate all of them, if done separately.

3.3 Intersection

Much as we computed set union in Section 3.1, we also want to compute set intersection. The way to do this is to exploit De Morgan's laws, namely by computing $A \cap B$ as $\neg(\neg A \cup \neg B)$.

For this, we merely utilise the techniques developed in the previous sections, noting that computing $\neg A$ and $\neg B$ can be done in parallel. Hence, the algorithm requires only two rounds of negation.

If the intersection $A \cap B$ is to be the final result returned by the algorithm (as in the first $\langle a, b, c \rangle$ example given in Section 2.2), it is, in fact, possible to simply compute

$$\neg(A \cap B) = \neg A \cup \neg B$$

instead, and then read it directly (as detailed in [5]), saving the second negation round altogether.

3.4 General Boolean functions

Because the Boolean operations described so far span the set of all Boolean functions, it is straightforward from here to implement any such function.

Specifically, Boolean functions can be represented in disjunctive normal form, and computing them directly in this form using the negation, disjunction and conjunction operators already described can always be done, requiring at most two negation rounds for any such function.

Moreover, if multiple functions need to be computed together, this can also be done in parallel, still in a grand total of at most two negation rounds.

4 DIFFERENTIAL PRIVACY

The process of adding noise, exemplified in Line 4 of Algorithm 2, entails the addition of a random number x_0 of "fake" accounts that are associated with a zero value and a random number x_1 of "fake" accounts that are associated with a nonzero value. These are "fake" in the sense that they are not in V and are eliminated later, on Line 32 of the same algorithm.

The question is therefore what distribution to choose x_0 and x_1 from, in order to ensure that results are privacy-preserving. Here, "privacy preserving" means that the value associated with any particular account should not be information that can be gleaned by the recipient of the noise-added information. Specifically, even without the addition of noise the FIU only learns in negation operations the total number of zero and the total number of nonzero tag values in the negated tag. The distribution of x_0 and x_1 needs to be chosen so as to ensure that the FIU cannot use these two revealed numbers in order to derive information about any specific tag value.

In this section, we describe distributions that satisfy this constraint.

4.1 FinTracer's noise formula

In [5], an algorithm $\mathcal{A}_{\epsilon,\delta}$ was described that generates a nonnegative number x from a distribution $D_{\epsilon,\delta}$ regarding which it was proved that it is the minimal-expectation distribution such that adding x fake items to a list provides a privacy preservation condition the paper defined as *strict* (ϵ, δ)*-differential privacy* for the identity of the elements of the list, from an observer that can only determine the size of the list after the addition of the fake items. Strict (ϵ, δ)-differential privacy is a strengthening of the conditions of standard (ϵ, δ)-differential privacy, which is defined as follows.

Definition 4.1. A probabilistic algorithm $\mathcal{A}$ working on an input X that is a set is called (ϵ, δ)*-differentially private* if for all X, with probability at least $1 - \delta$ over the execution probabilities of $\mathcal{A}$, the output v of the algorithm satisfies that for any other input set Y, with $|X \triangle Y| = 1$ (where $\triangle$ is the symmetric set difference operator),

$$e^{-\epsilon}\text{Prob}[\mathcal{A}(Y) = v] \leq \text{Prob}[\mathcal{A}(X) = v] \leq e^{\epsilon}\text{Prob}[\mathcal{A}(Y) = v].$$

In this paper, we extend the work of [5] to describe how to use Algorithm $\mathcal{A}_{\epsilon,\delta}$ inside a larger algorithmic framework that allows users to perform operations on tags, that guarantees an overall (ϵ_0, δ_0)-differential privacy for the entire system, for any chosen ϵ_0 and δ_0.

To do this, we utilise the following property.

Theorem 3 (Additivity of differential privacy). *If $\mathcal{B}$ is an algorithm whose output on input X is informationally-equivalent to $(\mathcal{A}_1(X), \ldots, \mathcal{A}_n(X))$, where each $\mathcal{A}_i$ is an (ϵ_i, δ_i)-differentially private algorithm, and where the execution of each $\mathcal{A}_i$ uses randomness that is independent of the randomness used for any other $\mathcal{A}_j$, then $\mathcal{B}$ satisfies $\left(\sum_{i=1}^{n} \epsilon_i, \sum_{i=1}^{n} \delta_i\right)$-differential privacy.*

Proof. Theorem 3 can be derived directly from Definition 4.1. The probability that none of the events with probabilities $\delta_1, \ldots, \delta_n$ will happen is bounded by $1 - \sum_{i=1}^{n} \delta_i$. If this happens, then the probability for each of the n $\mathcal{A}_i(X)$ results is bound to within a multiplicative range of $e^{\pm\epsilon_i}$ compared to the respective $\mathcal{A}_i(Y)$. Hence, the overall probability of the result is bound to within $\prod_{i=1}^{n} e^{\epsilon_i} = \exp(\sum_{i=1}^{n} \epsilon_i)$. □

4.2 Resilience against multiple changes

Definition 4.1 is limited in that it allows the addition or subtraction of only a single element in X. In conventional ϵ-differential privacy, this is not an issue, because resilience against multiple changes in X can be derived by transitivity, but for (ϵ, δ)-differential privacy this is not so.

We therefore define the following, to describe resilience against m changes.

Definition 4.2. A probabilistic algorithm $\mathcal{A}$ working on an input X that is a set is called (ϵ, δ, m)*-differentially private* if for all X and for all integers k in $1 \leq k \leq m$, with probability at least $1 - k\delta/m$ over the execution probabilities of $\mathcal{A}$, the output v of the algorithm satisfies that for any other input set Y, with $|X \triangle Y| \leq k$ (where $\triangle$ is the symmetric set difference operator),

$$\begin{aligned} e^{-k\epsilon/m}\text{Prob}[\mathcal{A}(Y) = v] &\leq \text{Prob}[\mathcal{A}(X) = v] \\ &\leq e^{k\epsilon/m}\text{Prob}[\mathcal{A}(Y) = v]. \end{aligned}$$

Definition 4.3. A probabilistic algorithm $\mathcal{A}$ working on an input set X and returning an integer $v \geq |X|$ is called *strict* (ϵ, δ, m)*-differentially private* if it is (ϵ, δ, m)-differentially private, and for all $v > |X|$ and all input sets Y, with $|X \triangle Y| = 1$,

$$\begin{aligned} e^{-\epsilon/m}\text{Prob}[\mathcal{A}(Y) = v] &\leq \text{Prob}[\mathcal{A}(X) = v] \\ &\leq e^{\epsilon/m}\text{Prob}[\mathcal{A}(Y) = v]. \end{aligned}$$

The reason we need to use (ϵ, δ)-differential privacy (and by extension (ϵ, δ, m)-differential privacy) instead of the more common ϵ-differential privacy is that our purpose is to add fake elements to the tag representing the result set, so as to obscure the exact size of its original support. To ensure that the revealed value of the final size of the set satisfies the stricter and more popular condition of ϵ-differential privacy, this value, v, would have had to have a positive probability to take any integer value. In our case, however, this size is restricted to be at least $|X|$, the size of the original set, because we are only adding fake entries, not removing any real entries.

In particular, ϵ-differential privacy will necessarily be broken whenever $v = |X|$, and no fake entries are added to X at all.

Instead, (ϵ, δ)-differential privacy ensures that ϵ-differential privacy is maintained in all but δ-rare cases.

The definition of strict (ϵ, δ)-differential privacy (and by extension strict (ϵ, δ, m)-differential privacy) requires that the stricter conditions of ϵ-differential privacy is met in all but the single case $v = |X|$ where, as discussed, ϵ-differential privacy is impossible to satisfy.

Consider, now, an algorithm $\mathcal{A}^m_{\epsilon,\delta}$ the generates a nonnegative integer value Δ out of a distribution $D^m_{\epsilon,\delta}$, and let $\mathcal{B}$ be an algorithm providing strict (ϵ, δ, m)-differential privacy to the size of its input set X by generating a value $\Delta \sim D^m_{\epsilon,\delta}$ and reporting $|X| + \Delta$.

Consider now the properties that such a distribution $D^m_{\epsilon,\delta}$ must satisfy in order for it to provide the stipulated strict (ϵ, δ, m)-differential privacy.

Let $p_i = \text{Prob}[\Delta = i]$. The value of p_i is by definition zero for all $i < 0$. After this, every p_i satisfies

$$\frac{p_{i+1}}{p_i} \geq e^{-\epsilon/m}.$$

This can be seen by substituting $k = 1$ in Definition 4.2.

Any such distribution is necessarily a weighted average of shifted exponential distributions, $E^{\epsilon/m}_i$, where the probability for $x \sim E^{\epsilon/m}_i$ to be any value $x \geq i$ is proportional to $\exp\left(-\frac{\epsilon}{m}(x - i)\right)$ and zero otherwise. Let α_i be such that $\alpha_i/(1-e^{-\epsilon/m})$ is the weight of $E_i^{\epsilon/m}$ in the representation of $D^m_{\epsilon,\delta}$ as a weighted mixture of all $E_i^{\epsilon/m}$ shifted geometric distributions.

The value of α_i corresponds to the size of the maximum contribution of $E_i^{\epsilon/m}$ to any p_i.

In [5], it was proved that for $m = 1$, the minimal-expectation distribution that satisfies strict (ϵ, δ, m)-differential privacy can be derived by a greedy algorithm that maximises each α_i in order. For a general m, this is no longer guaranteed. However, we will still use the greedy algorithm. The distribution $D^m_{\epsilon,\delta}$ derived through such greedy optimisation is described in Algorithm 3.

In Algorithm 3, we use p_i, as before, to represent Prob$[x = i]$ for $x \sim D^m_{\epsilon,\delta}$.

Algorithm 3 Computing $D^m_{\epsilon,\delta}$

```
1: $p_{-1} \leftarrow 0$.    ▷ $x = -1$ is not in the support, but $p_{-1}$ is here for convenience.
2: for $i = 0, 1, \dots$ do
3:     if $i < m$ then
4:         $\alpha_i \leftarrow \min\left(\delta/m - p_{i-1}e^{-\epsilon/m}, 1 - e^{-\epsilon/m} - \sum_{j=0}^{i-1}\alpha_j\right)$.
5:     else
6:         $\alpha_i \leftarrow \min\left(p_{i-1}(e^{\epsilon/m} - e^{-\epsilon/m}), 1 - e^{-\epsilon/m} - \sum_{j=0}^{i-1}\alpha_j\right)$.
7:     end if
8:     $p_i = p_{i-1}e^{-\epsilon/m} + \alpha_i$.
9: end for
```

The basic idea for Algorithm 3 is that each α_i is maximised sequentially, subject to the constraints imposed on it by the definition of strict (ϵ, δ, m)-differential privacy. Namely:

(1) The weight of each shifted geometric distribution in the linear combination is $\alpha_i/(1-e^{-\epsilon/m})$, for which reason the α_i must sum up to $1 - e^{-\epsilon/m}$.
(2) For $i < m$, $\sum_{j=0}^{i-1} p_j$ cannot exceed $i\delta/m$, for which reason greedy attribution limits each p_j in this range to δ/m.
(3) For $i \geq m$, p_i must be between $e^{-\epsilon/m}p_{i-1}$ and $e^{\epsilon/m}p_{i-1}$, to satisfy general ϵ/m-differential privacy.

If ϵ is sufficiently small, the first few α_i will be constrained by δ/m, the next few by $p_{i-1}(e^{\epsilon/m} - e^{-\epsilon/m})$, one additional α_j will be constrained by $1 - e^{-\epsilon/m} - \sum_{j=0}^{i-1}\alpha_j$, and all later α_j will be zeroes.

For larger ϵ values, the number of α_i in either of the first two categories may be zero.

This insight regarding the α_i values also allows computing each p_i using a closed-form formula, as follows.

Let $k = \max\left(0, \left\lceil \frac{m}{\delta} - \frac{1}{1-e^{-\epsilon/m}} \right\rceil\right)$.

For $0 \leq i < \min(k, m-1)$, $p_i = \delta/m$.

If $k < m$, for $i \geq k$, $p_i = \left(1 - e^{-\epsilon/m}\right)\left(1 - \frac{k\delta}{m}\right) e^{-\epsilon(i-k)/m}$.

Otherwise, for $i \geq m-1$, $p_i = \left(1 - \frac{(m-1)\delta}{m}\right) p^{\epsilon/m, \delta/(m-(m-1)\delta)}_{i-(m-1)}$,

where $p_i^{\epsilon,\delta}$ is the probability that $x \sim D_{\epsilon,\delta}$ equals i. Such a variable can be generated using $\mathcal{A}_{\epsilon,\delta}$, whose code was given in [5].

The closed form formula, in turn, allows one to generate a variable $x \sim D^m_{\epsilon,\delta}$ as described in Algorithm 4.

Algorithm 4 Generating variable $x \sim D^m_{\epsilon,\delta}$

```
1: $k \leftarrow \max\left(0, \left\lceil \frac{m}{\delta} - \frac{1}{1-e^{-\epsilon/m}} \right\rceil\right)$.
2: if $\min(k, m-1) > 0$ then
3:     $\rho \leftarrow$ Bernoulli($\min(k, m-1)\delta/m$).
4: else
5:     $\rho \leftarrow 0$.
6: end if
7: if $\rho = 1$ then    ▷ Happens with probability $\min(k, m-1)\delta/m$.
8:     $x \leftarrow$ UniformInteger($0, \min(k, m-1) - 1$).
9:                 ▷ Uniformly distributed between bounds, inclusive.
10:    return $x$.
11: else if $k < m$ then
12:    $y \leftarrow$ Geometric($e^{-\epsilon/m}$).
13:    return $y + k$
14: else
15:    Generate $y$ from $D_{\epsilon/m, \delta/(m-(m-1)\delta)}$ using Algorithm $\mathcal{A}_{\epsilon,\delta}$.
16:    return $y + m - 1$.
17: end if
```

4.3 Origin tracking

The above showed how we can attain (ϵ, δ, m)-differential privacy for a given instance of noise addition, over our choice of ϵ, δ and m. In real usage, however, we would like to set our differential privacy requirements once, system wide, and have the individual choices of ϵ, δ and m for each individual operation automatically computed so as to uphold the overall desired system-wide conditions.

Our solution to providing system-wide differential privacy requires each tag to be associated with the list of its *origins*. Each such "origin" designates a set of accounts whose identity needs to be kept private from the FIU. A tag's origin list is the set of all origins whose information had been used in constructing the tag's values.

Specifically:

(1) Whenever the user defines a new tag by means of a query (such as when using the queries Q_S and Q_D), this creates a new origin and assigns this new origin as the sole origin for the newly created tag.
(2) When computing a new tag from existing tags, the new tag's set of origins is the union of the sets of origins of all inputs to the computation. For example, if we compute the tag intersection "$B \leftarrow B_1 \cap B_2$", the origin list for B will be the union of the origin lists of the two operands B_1 and B_2.

Origins are thus, mathematically, the private account sets, so their values are by definition unknown to the FIU, but the FIU can nevertheless catalogue origins, such as by associating them with the queries that originally defined them, and the time at which those queries were run.

Neither the mathematical description of the origin as a private account set, nor the information by which an origin can be catalogued is, however, the information programmatically retained (both by the FIU and by each $f \in F$ separately) on each such origin, in order to perform differential privacy calculations. This calculus uses only two real nonnegative numbers, ϵ_s and δ_s, for each origin s. Upon creation of an origin, these numbers are initialised by the values ϵ_0 and δ_0, respectively. We refer to ϵ_0 and δ_0 as the origin's *initial privacy budget*, and to the current values of ϵ_s and δ_s at any given time as the origin's *remaining privacy budget*.

Information that is known to the FIU does not create a new origin, even if it is stored in the form of a tag, because such information does not need protection from the FIU. For example, if the FIU has a given list of accounts, explicitly numbered, which is of interest to a particular investigation (e.g., accounts associated with known persons of interest), such a list would not form its own, new origin, even if used to initialise a tag.

Whenever an operation is performed on a tag that requires differential privacy noise, this operation has a cost: it reduces the remaining differential privacy budgets of each of the tag's origins. Each origin s has some value deducted from its remaining ϵ_s and δ_s budgets. Here, the system offers a trade-off: we can set (e.g., at user's choice) how much privacy budget is to be expended for the operation; the more differential noise is added, the smaller the spent differential privacy budget.

Figure 2 presents an example.

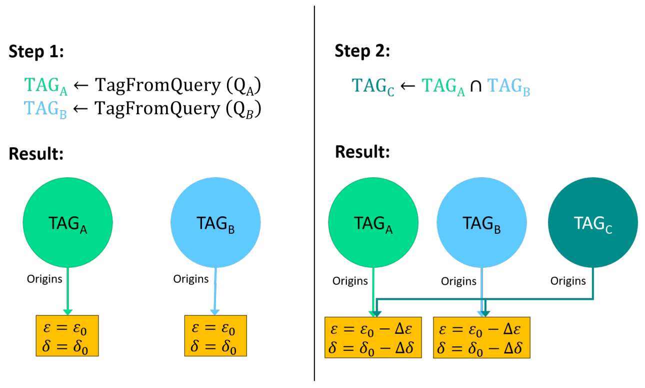
Step 1:
$TAG_A \leftarrow$ TagFromQuery (Q_A)
$TAG_B \leftarrow$ TagFromQuery (Q_B)
Result:
TAG_A
TAG_B
Origins
Origins
$\varepsilon = \varepsilon_0$ $\delta = \delta_0$
$\varepsilon = \varepsilon_0$ $\delta = \delta_0$
Step 2:
$TAG_C \leftarrow TAG_A \cap TAG_B$
Result:
TAG_A
TAG_B
TAG_C
Origins
Origins
Origins
$\varepsilon = \varepsilon_0 - \Delta\varepsilon$ $\delta = \delta_0 - \Delta\delta$
$\varepsilon = \varepsilon_0 - \Delta\varepsilon$ $\delta = \delta_0 - \Delta\delta$

Figure 2: An example of how differential privacy budget is managed. In "Step 1", two tags are defined from a query. Each is associated with its own new origin, each origin set to an (ϵ_0, δ_0) initial differential privacy budget. In "Step 2", a new tag is computed from the two existing tags. Its origin list is the union of the origin lists of both original tags. However, the origins in this list will have lost some of their differential privacy budget in the process of generating the new tag. How much is lost depends on the FIU user, with less budget expenditure corresponding to the addition of more differential privacy noise in the process of computing the new tag. The values of (ϵ, δ) for each origin are known to the FIU user, but not the contents of any of the three tags.

Consider an operation that is performed simultaneously over k tags. (For example, this can be a simultaneous negation of multiple tags.) Each tag is associated with a list of origins. Some origins may be unique to only one of the k tags, others may contribute to more tags, and potentially to all k. Let K be the maximum number of tags associated with any origin used in the computation.

Now, let us suppose that we use Algorithm 4 to generate the number of fake accounts used according to distribution $D^K_{\epsilon,\delta}$. We will for this require two independent variables $x_0, x_1 \sim D^K_{\epsilon,\delta}$, where x_0 signifies the number of fake accounts with zero tag value and x_1 signifies the number of fake accounts with nonzero tag value.

We use the following algorithm, which we will name $\mathcal{B}$, to determine what (ϵ, δ) combinations are allowed.

(1) Every origin defined in the system receives an initial differential privacy budget (ϵ_0, δ_0).
(2) At each operation in which tag values are communicated to the FIU, each origin that contributes to k' of the tags thus communicated has its differential privacy budget reduced by $(2\epsilon k'/K, 2\delta k'/K)$, where K is the maximum number of tags associated with any origin used in the computation and $D^K_{\epsilon,\delta}$ is the distribution from which the number of fake zero-valued tags x_0 and the number of fake nonzero-valued tags x_1 has been generated.
(3) The values of ϵ and δ in each operation are bound such that at the end of each operation the differential privacy budget of all origins remains nonnegative.

The choice of an appropriate (ϵ, δ) pair is made at the FIU node. One possible design for the system, for example, may allow this choice to be made by the querying user. In this case, the software should ensure that all origins participating in the operation have enough differential-privacy budget remaining, before the operation can be carried out. (In a practical implementation, it is also useful for the software to monitor the expected amount of fake items that are to be inserted due to a given operation, and to throttle operations for which this expectation is unreasonably high.)

If the system is meant for ongoing operation by intelligence analysts, it is prudent to always allow for the possibility that origins will need to be queried again. In this case, instead of requiring that the differential privacy budget at the end of each operation will be nonnegative, the algorithm should require it to remain positive.

If one does not wish to allow individual querying users to set their own differential privacy budget expenditures for each operation, a convenient way to set this level of expenditure automatically is, for example, to allot for each operation half (or some other set portion) of the maximum allowed.

Theorem 4. *Let $S_1, \ldots, S_t$ be the set of origins defined in the system, and let $a_1, \ldots, a_s$ be the set of all accounts in the system. In this context, we identify each origin with the set of results returned by the query associated with that origin. Thus, each of $S_1, \ldots, S_t$ is a subset of $\{a_1, \ldots, a_s\}$.*

Define M_{ij} as the matrix such that $M_{ij} = 1$ if $a_i \in S_j$ and zero otherwise. If all entries in M_{ij} are independent, then algorithm $\mathcal{B}$ ensures (ϵ_0, δ_0)-differential privacy to all S_j under all Boolean operations.

Proof. Consider a system, as above, that has undergone some number, R, of Boolean operations, $O_1, \ldots, O_R$. (For our purpose, we can consider only the underlying negation operations, as these are the only operations that reveal any information.) During each of these operations, two numbers are revealed to the FIU, these being the number of zero and the number of nonzero tag values communicated to the FIU. As per Theorem 2, the algorithms used for Boolean operations ensure that no other information is revealed. Thus, the total amount of information revealed to the FIU is these $x^0_1, \ldots, x^0_R$ and $x^1_1, \ldots, x^1_R$ values.

Proving (ϵ_0, δ_0)-differential privacy as per the theorem requires us to show that the viewed values $\{x^0_1, \ldots, x^0_R, x^1_1, \ldots, x^1_R\}$ have, except with probability at most δ_0, the same probability to be produced by all Boolean operations, up to a multiplicative difference of at most e^{ϵ_0}, as the probability for them to be produced by the same Boolean operations had one entry in M been flipped.

Let us say that M_{ij} had been flipped, and consider, accordingly, only the differential privacy afforded to the origin S_j (given that, by construction, there was no change to any other origin value).

For every r in the range $1 \leq r \leq R$, the values of x_r^0 and x_r^1 are chosen in $\mathcal{B}$ so as to satisfy the conditions of $(\epsilon_r, \delta_r, K_r)$-differential privacy, for user-chosen ϵ_r and δ_r, where $K_r = \max_{j'} \left(k_r^{j'}\right)$ and where k_r^j is the number of tags simultaneously negated at operation O_r that have S_j as part of their origin list.

By definition, we therefore have $K_r \geq k_r^j$, and can use Definition 4.2 to show that the distribution of no x_r^q can change more than is allowed by $(\epsilon_r k_r^j / K_r, \delta_r k_r^j / K_r)$-differential privacy between any input sets X and Y that differ by at most k_r^j elements. Here, "input set" refers to the intermediate data sent to the FIU as part of Algorithm 2, before the addition of any noise.

In other words, if a change of a single value in S_j cannot propagate to a change of more than k_r^j elements in X_r, the intermediate data sent to the FIU while running operation O_r (before the addition of any noise) then the value of x_r^q satisfies $(\epsilon_r k_r^j / K_r, \delta_r k_r^j / K_r)$-differential privacy for the set S_j.

Because all x_r^q are visible to the FIU, and because the random bits generating each are independent, we can use Theorem 3 to determine that the overall differential privacy protection provided by the system to S_j is the sum of all such $(\epsilon_r k_r^j / K_r, \delta_r k_r^j / K_r)$ values. In particular, by summing over both x_r^0 and x_r^1 we get $(2\epsilon_r k_r^j / K_r, 2\delta_r k_r^j / K_r)$, and the total value is the sum of this over all r.

By construction, Algorithm $\mathcal{B}$ ensures that this sum is always under (ϵ_0, δ_0), thus upholding the desired level of differential privacy protection to all S_j, as stipulated by the theorem.

It remains to show, therefore, only that a change of a single value in S_j cannot propagate to a change of more than k_r^j elements in X_r.

The value of X_r is the permuted, sanitised and refreshed concatenation of all values in all tags negated in O_r. Because in Boolean operations the computation of an account's tag value only relates to the value of the same account in the origin tags, it ultimately depends only on whether or not the account participates in any given origin. Specifically, a change in account a_i in S_j can only impact one tag value, that of a_i, in any given tag, and even then only in tags for which S_j is an origin. Thus, a single change in S_j can propagate to at most k_r^j elements in X_r by the definition of k_r^j, and therefore the theorem is proved. □

For a discussion of the limitations of the guarantees afforded by Theorem 4, see Appendix B.

5 PERFORMANCE

We tested the behaviour of the system against real Australian transaction data, kindly provided by AUSTRAC, in order to validate that the system is able to detect actual crime typologies and actual instances of financial crime in the real dataset. However, for the purpose of measuring the system's ability to scale out (as well as to provide results that will be reproducible without access to classified data), we also ran the system on synthetic, random data.

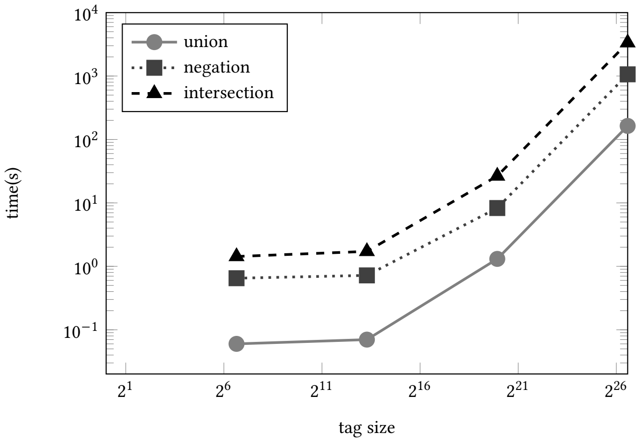

Figure 3: Mean serial execution time vs tag size for union, negation and intersection operations.

The number of customers served by each of Australia's four major financial institutions is approximately 15 million [12, 14], or roughly 2^{26} customers across all Australian banks. We therefore estimate $|V|$ at 2^{27}, for an average of 2 accounts per customer. We generated random tags of sizes scaling up to this $|V|$ and ran union, negation and intersection 5 times each on every tag size tested, to demonstrate the system's consistent performance in each case.

Tables 1, 2 and 3 list the timing results for each operation. The numbers reflect the "serial time" of the operations, this being the execution time required by the FIU node plus the maximum execution time among all other nodes. The mean values for the serial execution time are plotted in Figure 3.

Run times were measured on a distributed deployment on the AWS cloud. Each of Australia's four major banks was represented in the experiment by a single EC2 `g4dn.12xlarge` node. These nodes were chosen due to their large memory sizes (192GB each) as well as having a GPU. The EC2 nodes used all have 48 vCPUs and 4 NVIDIA T4 16GB GPUs. The reported timing experiments, however, only use a single vCPU and a single GPU per node to execute the computations described in this paper.[4] Both the CPU and the GPU parts of our algorithms are straightforward to parallelise, but such parallelisation has been left for future work.

The algorithms were implemented in Go, running on the available CPUs, except for Curve25519 operations, which were implemented in Cuda C++ and executed on the available GPUs.

The results show that most operations require mere seconds, and even the heaviest operations when run at full scale (and using only one CPU and one GPU at each node) complete in less than an hour.

Regarding communication sizes, in our implementation ciphertexts are transported between nodes in their extended projective representation [9]. At 256 bytes per ciphertext, this translates to a total of 64GB communicated for each negation at $|V| = 2^{27}$. This

[4]Some non-computational tasks such as inter-node communication run in parallel across multiple vCPUs.

Table 1: Union times (seconds).

$\|tagA\|$ and $\|tagB\|$	min	max	mean	std	relative std
100	0.06	0.06	0.06	0.00	0.01
10000	0.07	0.07	0.07	0.00	0.01
1000000	1.28	1.34	1.31	0.02	0.01
100000000	162.29	167.04	163.46	2.01	0.01

Table 2: Negation times (seconds).

$\|tag\|$	min	max	mean	std	relative std
100	0.62	0.68	0.65	0.02	0.03
10000	0.69	0.75	0.72	0.02	0.03
1000000	8.18	8.37	8.30	0.07	0.00
100000000	1059.03	1068.04	1063.62	4.19	0.00

Table 3: Intersection times (seconds).

$\|tagA\|$ and $\|tagB\|$	min	max	mean	std	relative std
100	1.40	1.44	1.43	0.01	0.01
10000	1.61	1.77	1.72	0.06	0.03
1000000	26.35	26.69	26.54	0.14	0.00
100000000	3324.74	3358.96	3345.68	13.86	0.00

could have easily been compressible by a factor of 4 by converting the ciphertexts to a more compressed representation before communicating them, at negligible addition to the computation times.

6 CONCLUSIONS

The ability to perform general local Boolean computations expands the spectrum of what a FinTracer-based system can compute obliviously from linear operations alone to the entire gamut of all that a vertex-centric program [10] running over the transaction graph can compute in $O(1)$. We have shown by the examples of in-between accounts, the computation of distances, describing complex typologies and chaining operations that even a small number of such Boolean operations can describe many complex relationships that are meaningful to a financial investigator. Using classified data made available by AUSTRAC for this research, we have shown that these tools can detect actual instances of financial crime in a real dataset.

Even when run at the scale of Australia's national economy, our algorithms were shown to be not only practical in both communication amounts and computation speeds, but also allowing for interactive work, as well as for the use of our nonlinear operations as building blocks in algorithms that are more complex and more semantically meaningful to investigators of financial crime, thus enabling, in a privacy-preserving way, investigation avenues that were previously impossible, impractical, or requiring a high cost in the privacy of uninvolved and innocent account-holders.

ACKNOWLEDGMENTS

This work was funded by AUSTRAC.

REFERENCES

[1] AUSTRAC. 2021. REGULATORY QUICK GUIDE: Tipping Off. https://www.austrac.gov.au/sites/default/files/2021-06/Quick%20guide%20-%20Tipping%20off.pdf.
[2] AUSTRAC. 2022. AUSTRAC Overview. https://www.austrac.gov.au/about-us/austrac-overview.
[3] Daniel J Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*. Springer, 207–228.
[4] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. 2008. Twisted Edwards curves. In *International Conference on Cryptology in Africa*. Springer, 389–405.
[5] Michael Brand, Hamish Ivey-Law, and Tania Churchill. 2023. FinTracer: A privacy-preserving mechanism for tracing electronic money. *Cryptology ePrint Archive* (2023).
[6] United Nations Statistics Division. 2021. United Nations: National Accounts Main Aggregates Database. https://unstats.un.org/UNSD/snaama/Index. Data upload: December 2021.
[7] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.
[8] Australian Government. 2021. Anti-Money Laundering and Counter-Terrorism Financing Act 2006. https://www.legislation.gov.au/Details/C2021C00243. Registered 30 June 2021.
[9] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. 2008. Twisted Edwards curves revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 326–343.
[10] Robert Ryan McCune, Tim Weninger, and Greg Madey. 2015. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Computing Surveys (CSUR)* 48, 2 (2015), 1–39.
[11] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. 2018. *Chapter 8.4 ElGamal public key encryption.* CRC press, 283–319.
[12] House of Representatives. 2019. Australia's four major banks and other financial institutions: Four major banks. https://parlinfo.aph.gov.au/parlInfo/download/committees/commrep/86eef00b-f8cd-4c13-b8c3-3cef597b3142/toc_pdf/Standing%20Committee%20on%20Economics_2019_11_08_7339_Official.pdf;fileType=application%2Fpdf#search=%22committees/commrep/86eef00b-f8cd-4c13-b8c3-3cef597b3142/0000%22. In *Standing Committee on Economics*, Nov 8 2019.
[13] Alex Sangers, Maran van Heesch, Thomas Attema, Thijs Veugen, Mark Wiggerman, Jan Veldsink, Oscar Bloemen, and Daniël Worm. 2019. Secure multiparty PageRank algorithm for collaborative fraud detection. In *International Conference*

on Financial Cryptography and Data Security. Springer, 605–623.

[14] Statista. 2022. Number of customers of major banks Australia 2020. https://www.statista.com/statistics/1211679/australia-big-four-banks-number-of-customers/. Published 2 Apr 2022.

A REFRESHING AFTER SANITISATION

In Algorithm 2, multiple tools are used to remove information from a tag prior to sending the remaining information to the FIU. Among these are "refreshing" and "sanitisation".

Refreshing and sanitisation provide different kinds of information erasure, which are useful in different contexts. To explain why they are needed, and why they are needed jointly, let us begin with a quick recap of ElGamal encryption.

When encrypting a plaintext, x, this being an integer modulo the elliptic curve group size, we first transform it into a group element, a process known as *encoding*. The result is $X = G^x$, where G is a generator of the group. We will throughout use lowercase letters to signify modular integers and uppercase letters to signify group elements, as was done here.

To encrypt a message m using a private key a and a corresponding public key $A = G^a$, we first encode m as $M = G^m$, then generate a random modular integer x, known as a *nonce*. The ultimate ciphertext is $(MA^x, G^x) = (MG^{ax}, G^x)$.

This definition inherently makes the ElGamal encryption scheme reliant on the hardness of two problems:

- First, it should be cryptographically hard to compute x from G^x. This is the *discrete log* problem. If computing the discrete log is possible for an adversary, the private key a can immediately be derived from the public key A.
- Second, it should be cryptographically hard to compute G^{ax} from G^a and G^x. This is the *computational Diffie-Hellman assumption.* This assumption is needed because both G^a and G^x are available to an attacker (the former being the public key and the latter being a given part of the ciphertext), and knowledge of G^{ax} allows retrieval of M. For true cryptographic security, one requires the stronger version of this assumption, the *decisional Diffie-Hellman assumption*, which is that this problem is hard even as a decision problem: given G^a, G^x and Y, it should be cryptographically hard to determine whether $Y = G^{ax}$.

For convenience, we write ciphertext tuples in additive notation. This is to say, instead of writing the ciphertext as $(MG^{ax}, G^x) = (G^{m+ax}, G^x)$, we will write $[m + ax, x]$, where the square brackets signify that all tuple elements are encoded into the elliptic curve group.

The "refresh" operation for ElGamal is an operation that takes a ciphertext $[m + ax, x]$ and returns a new ciphertext $[m + ax', x']$, with an independent, uniformly distributed x', thus disallowing any observer of both $[m + ax, x]$ and $[m + ax', x']$ who does not have the private key from being able to determine whether the two ciphertexts carry the same encrypted message.

A completely different type of information protection is afforded by sanitisation, which replaces $[m+ax, x]$ by $[km+akx, kx]$, where k is an independent, uniformly-distributed nonzero integer modulo the group size.

The purpose of sanitisation is to erase from m all information other than whether $m = 0$ or $m \neq 0$. A zero remains a zero in the multiplication, whereas multiplying any nonzero value by k results in a new ciphertext encrypting a message that is uniformly random among all possible nonzero values.

In Algorithm 2, tags are both refreshed and sanitised before being sent to the FIU. The reason for this is that the information erasure provided by each operation is unrelated. Sanitisation is required because FinTracer tags that haven't been sanitised may carry information that should not be sent to the FIU. For example, in Algorithm 2 the FIU is allowed, on lines 19 and 21, to set tag values explicitly, so it is possible that a user may encode more information in them than just zeroes and ones. We sanitise to avoid the potential for any such information leak.

However, even given sanitisation, refresh is still required. To see why, consider a situation where a tag is sent from the FIU to a financial institution and then returned without refresh from the financial institution to the FIU without visiting other financial institutions.

When the FIU sends the tag $[m + ax, x]$, it knows all of a, m and x. We will consider the case $m \neq 0$, as in the case $m = 0$ sanitisation and refreshing are identical.

Now, the FIU receives from the financial institution a tag $[m' + ax', x']$, which it suspects to be the product of sanitisation from the originally sent tag. To determine this, the FIU needs to answer the question of whether there exists a k value such that $m' = mk$ and $x' = xk$.

The received message contains $G^{x'}$, and by decrypting with the private key a, the FIU can also determine from it $G^{m'}$.

The FIU can now check whether the new tag is a sanitisation of the old tag by testing whether $\left(G^{m'}\right)^x$ equals $\left(G^{x'}\right)^m$.

This process can also be used to check whether the returned tag was a sanitised version of the sum of some specific set of tags that were sent by the FIU, and the FIU can enumerate over such possibilities, if their overall number is not too high.

In summary, for a party that has the private key and can also gain knowledge of the nonce's original integer value, x, sanitisation does not perform the information erasure that refreshing does, so tags sent to it must be both sanitised and refreshed beforehand.

This is different to the situation of sending a tag to a party that does not have the private key, where only refreshing is required. For such a party to determine whether a tag equals its refreshed counterpart is equivalent to a decisional Diffie-Hellman problem.

B DIFFERENTIAL PRIVACY LIMITATIONS

Throughout this paper, we discuss the privacy guarantees that the described method for performing Boolean operations provides. For the most part, these guarantees are afforded cryptographic protection by the protocols employed, and in such cases we have proved that the guarantees are upheld.

In Section 4.3, however, we introduce Algorithm $\mathcal{B}$, which provides a different kind of protection, namely by use of differential privacy. Theorem 4 provides formal guarantees for the use of Algorithm $\mathcal{B}$. However, unlike other forms of protection discussed in this paper, the guarantees provided by Theorem 4 are

(1) Quite limited, and also
(2) Contingent on certain assumptions.

Our guarantees are *limited* in that our use of differential privacy only protects the privacy of individual people from being associated with particular groups. (If S is a set of accounts defined by any criterion, it should not be possible for the FIU to ascertain whether a particular account is in S or not.) By contrast, our use of differential privacy does not afford the same protection to entire groups, nor do we attempt to obfuscate from the FIU the approximate size of the entire set S. Both of these may be sensitive (e.g., for commercial reasons), but they are not protected by the algorithm. Our guarantees are also limited by only covering Boolean operations, and not any other type of operation the user may run on the system.

Our guarantees are *contingent on assumptions* in that Theorem 4 requires independence in the data. Chiefly, we assume that whether any particular account belongs to the set defined by any origin is

(1) Independent of whether any other account is in the same set (the *account independence* assumption), and
(2) Independent of whether the same account belongs to the set defined by any other origin (the *origin independence* assumption).

Consider, now, a system as described, employed for the purpose of detecting financial crime. The veracity of the above assumptions strongly depends on how this system is used in practice.

In this appendix, we analyse the expected usage of such a system, determine to what extent such assumptions are justified, and discuss what actions operators of such a system should take in practice in order to ensure that the desired privacy guarantees are met.

Our analysis will show that common operational scenarios do tend to violate both account independence and origin independence, requiring such a system, in practice, to be constantly monitored for use and abuse.

Having said this, we note the following as mitigating factors.

First, any information reveal in our algorithms (excluding the reveal of any final result by the original FinTracer algorithm) comes from the transmission of data to the FIU at Step 10 of Algorithm 2. As per Theorem 2, this reveal is solely of the number of zero and the number of nonzero values communicated.

Second, in Algorithm 2, this transmission to the FIU is only used by the FIU in order to perform a local computation whose results are ultimately sent back to the financial institutions in Step 25. The recipients gain by this no new information, because they merely receive refreshed encrypted values in an amount that they, themselves have previously stipulated. Importantly, following this transmission the FIU itself no longer requires the data, and it is never required by any FIU-side user.

This being the case, we can make the following observations.

(1) Any information reveal that results from a failure of differential privacy guarantees is necessarily towards the FIU, not towards any financial institution, and so does not run afoul of any applicable law.
(2) The software system supporting such queries can be programmed so as not to display any of the intermediate information that is generated for the purpose of oblivious nonlinear computation, and to erase all such transient information as soon as the operation is done and the information is no longer needed. This would ensure that unless the software is actively tampered with on the FIU side, no information at all is leaked to any user, except for any final result explicitly requested by FIU users.

Given the above, the risks posed by the system can be classified as follows.

Intermediate Result Risk: There is the risk that FIU users will design queries where some of the intermediate results sent to the FIU
- Break the assumptions of Theorem 4, and
- Contain sensitive information, and
- Are exposed due to tampering of the FIU-side software.

Final Result Risk: There is the risk that FIU users will explicitly request from the system final results that, by themselves, are privacy sensitive.

In the first case, the entire gamut of available cybersecurity measures can be utilised to mitigate the system security aspect of the risk.

To mitigate the risk portions related to the behaviour of FIU users, which is relevant in both cases, one should make sure system users receive the appropriate training; also, the system's use should be monitored.

For monitoring external to the FIU, we note that the system runs a collaborative protocol: all financial institutions running the protocol are aware of what FinTracer queries and what Boolean functions are being computed, so the same monitoring can be done by all participants. It is only the results that are privacy-protected.

In this section, we describe aspects of standard-looking use that can cause our data assumptions to break. These are the usages that system operators should be trained to watch out for and to avoid as much as possible in the context of Intermediate Result Risk.

Mitigation against Final Result Risk is more straightforward:

(1) Operators should be trained only to retrieve accounts matching full criminal typologies, rather than any interim results, and
(2) The system should be programmed to automatically disallow some types of queries that are inherently privacy non-preserving. For example, the system may automatically block queries whose return results exceed a given size threshold. (It is even possible to set such a threshold stochastically, using an additional layer of differential privacy, so as to hide from the user even the exact size of the blocked result.)

B.1 Differential privacy and propagation

Theorem 4 deals only with Boolean operations. There is, however, little reason to use the system purely for Boolean computation. The system's strengths come from combining the Boolean computations described in this paper with the FinTracer algorithm, providing tag propagation through the underlying transaction graph, which was introduced in [5].

Tag propagation, however, voids the assumption that the values of tags are independent between accounts.

Consider a FinTracer tag x. This is a mapping from accounts to encrypted values. For mathematical convenience, let us take each account, a, and map it to a vector position $p(a)$, then $\tilde{x}$, defined by $\tilde{x}[p(a)] = x(a)$, can be thought of as a vector holding the same information as x. (This is, of course, a purely mathematical operation, not one performed during any part of the run of the algorithm: no party involved in the computation has enough information to construct the mapping p. In practice, each party is only aware of a small portion of the accounts in the domain of any tag.)

Under this mathematical model, FinTracer propagation of x to a new tag, y, is mathematically equivalent to $\tilde{y} = M\tilde{x}$, where M is the adjacency matrix of the graph over which propagation is done (a value known jointly to the computing parties) and $M\tilde{x}$ is matrix by vector multiplication. For convenience, we denote the relationship between x and y as $y = Mx$. We refer to M as the *propagation operator* of the FinTracer graph.

Given that M is a general matrix, there are relationships between x and y that cross single-account boundaries, and may cause a correlation between the values assigned to multiple accounts in y, contrary to our assumption of account independence.

The following is an example of how tag propagation, breaking the account independence assumption, can be used to leak private information from a FinTracer tag.

Consider a tag x. There is some set of accounts, S, for which the tag values in x are nonzero. We wish to ascertain whether a particular person's account, a, is in S. Differential privacy preservation essentially means that we shouldn't be able to determine this. However, we will show that this is possible to do if one has access to the total number of nonzero tag values in multiple queries, even after the introduction of differential privacy noise.

Let us define the set $A = \{a\}$, and a new tag x' whose positive set is $S' = S \setminus A$.

Differential privacy noise is intended to prevent a user from being able to determine whether there has been a change in any single element, so it should not be possible to tell whether S and S' are the same, which is equivalent to determining whether $a \in S$.

However, let $y = Mx$ and $y' = Mx'$ be the images of x and x', respectively, as mapped by some propagation operator M.

A difference of the single account a between x and x' can translate to an arbitrarily larger difference between y and y', depending on the out-degree of a in the FinTracer graph. Even if the FinTracer graph is bounded to have low degree, one can simply continue propagating along it, to further accentuate the difference to any desired level.

The same trick can also be applied when only a single tag is being queried. Consider the tag representing the set $S \cap A$. This is non-empty if and only if a is in S. If we merely propagate this tag through enough iterations, it may grow to any desired size, assuming it started off as non-empty. But if it started off empty, it will remain empty no matter how many times it is propagated, making the two cases easily distinguishable.

In terms of the design of the system, our recommended handling of this situation is to define the set underlying any tag that is a direct output of FinTracer propagation as a new "origin". The concern is that this new origin may be highly correlated with other origins in the system, which may break the assumptions of Theorem 4. To ensure that account independence is maintained as much as possible, FIU operators should be trained not to work with tags of unnecessarily small support or with pairs of tags that are unnecessarily similar.

B.2 Considerations when defining new origins

The assumption of origin independence is problematic in the context of defining new origins, i.e. when the FIU defines a set of accounts by use of a description that is sent out to financial institutions (e.g., in the form of a database query) for the financial institutions to resolve.

It is an inherent problem in all differential privacy schemes that if an attacker is given free access to query a system, by merely repeating the same query enough times they can erode the protections provided by the differential privacy noise simply due to the law of convergence to the mean.

Our algorithm protects against this on a technical level, by preventing the same origin from being overly reused, but if a user of the FIU node inadvertently redefines the same set through a second description, defining by this an ostensibly-unrelated new origin, this cannot be discovered purely using software: the same set can be described using many semantically-identical descriptions that software cannot equate.

There are certainly situations in which such unintentional reuse of origin definitions may occur in the standard course of operating a FinTracer-running system. A user trying to refine a query by iteratively improving it may, in the process, create origin definitions that are not identical but still quite similar, for example.

Alternatively, a new description may just happen to coincide with *some* past origin defined over the course of the usage of the system, as may easily happen for commonly useful sets.

In practical terms, it is impossible to maintain true origin independence in a system that is used over a long period of time and has many users. However, users can be trained to avoid such "iterative refinement", and to consult an available library of existing (or commonly-used) origins before defining new ones.

Once again, such training is never the sole line of defence for the system's differential privacy guarantees, but rather an additional defence, if all other precautions, such as cybersecurity measures preventing software tampering, have failed.