Oblivious setting of source accounts in FinTracer

Michael Brand

November 23, 2023

Executive Summary

We describe a method of defining a FinTracer tag from an FIU-known set of accounts that does not reveal information to the REs and does not reveal private information to the FIU.

1 Introduction

In the FinTracer algorithm, a key task is the setting up of a *tag*, where a tag is a dictionary mapping from account identifiers (BSBs and account numbers) to semi-homomorphically encrypted values. The tags are held by the reporting entities (REs), each RE's dictionary keys being accounts managed by that RE, and the semi-homomorphic encryption is such that its private key is held only by the FIU.

Typically, to start the process, a set of accounts is defined, and the tag encodes the choice of account set by mapping accounts in the set to nonzero plaintext values and accounts not in the set to zero values. Dictionary keys that map to zero values do not influence the computation and can at times be omitted altogether, but given that the values are encrypted and that the REs do not have the decryption key, the REs usually do not know which dictionary keys map to a zero and which not, so cannot perform this pruning.

In [2], two options are described for setting up a tag: one in the case in which the FIU is able to define the target set implicitly, by means of a description, one in the case in which the FIU has an explicit list of account identifiers. In this document, we focus on the latter case, where the FIU has an explicit list.

The algorithm described in [2] requires the FIU to provide the REs with the target list of accounts explicitly (each RE receiving only the part of the list containing their own accounts). This is useful for the (typical) case, where the FIU is happy to share the list explicitly with the REs. Consider, however, the situation where this list is classified and cannot be shared.

A simple fallback position is for the FIU to generate an entire tag, complete with as many account ID keys mapping to zeroes as is necessary, and to send *that* to the REs instead of merely those accounts mapping to nonzero.

This variant has the advantage that the identities of the accounts in the classified list are (per assumption) not revealed to the REs, but the cost is a different kind of information leakage: the FIU must gain familiarity with the REs' lists of account IDs, the vast majority of which will be innocent accounts.

In this document we describe a new algorithm that circumvents both problems. It allows the FIU to set up a tag based on an explicit list of accounts without the REs gaining any information regarding which accounts are in the set and without the FIU gaining privacy-protected information regarding innocent accounts.

This algorithm, specifically, does more than simply allow the FIU to define a set of accounts. It allows the FIU to generate an entire tag, complete with tag values that each account is mapped to.

We describe the algorithm in parts, first dealing with the "core" algorithm, this being the mechanism by which the FIU can set such a tag, then covering additional parts of the algorithm that function as safeguards, making sure that all parties are following the protocol and do not utilise it as a mechanism for gaining knowledge improperly - for example gaining knowledge of which accounts are held by the REs.

Any attempt to deviate from the protocol by either the FIU or the REs will immediately be detected by the other parties, triggering an alert. Such alerts, when detected by any party, should be propagated to all parties and cause execution of the protocol to be immediately aborted.

2 The algorithm

The following is a description of the algorithm, which we present in parts.

2.1 The core algorithm

Let A be the set of accounts, explicitly known to the FIU, for which the FIU wishes to generate a corresponding tag.

We generate the tag fragment related to each RE separately, so for the purpose of this description we can assume only two parties: the FIU and one RE.

We choose $\hat{A}$ to be a superset of A that is defined by a description (not as an explicit list of accounts) that can be shared with the REs. Such a set $\hat{A}$ can always be found because one can always choose as $\hat{A}$ the set of "all accounts". However, the costs associated with the algorithm are directly related to the size of $\hat{A}$ (not A), so if a smaller $\hat{A}$ can be picked, that is advantageous.

Note 2.1. When extrapolating to multiple REs, the description for $\hat{A}$ should be the same for all REs.

One item that the algorithm does not keep private is the approximate size of $\hat{A}$. This is communicated from the RE to the FIU in the form of a parameter, S, that serves as an upper bound for $|\hat{A}|$, and that determines the overall communication and computation costs of the algorithm. Section 3 discusses how to choose S so as to ensure that communicating the approximate size does not in itself reveal protected private information.

The core algorithm uses, additionally, two parameters, α and C. How these should be chosen is also discussed in Section 3, but, importantly, (α, C) is purely a function of S, so can be computed independently by both sides given S.

Specifically, the choice of (α, C) ensures the following.

Let $\mathcal{A}$ be the space of all potential account identifiers (e.g., the space of all 15-digit strings), let $S' = \lceil \alpha S \rceil$, let r be a random variable distributed uniformly in some space R, and let $H_r^1, \ldots, H_r^C$ be C hash functions, chosen randomly as a function of r, that map from $\mathcal{A}$ to the numbers in $[0, S')$. We assume that the space of hash functions is chosen such that over such a randomly-chosen r for any $a \in \mathcal{A}$ the value of each $H_r^c(a)$ is uniformly distributed in $[0, S')$ and independent of the value of all other $H_r^{c'}(a')$, for $1 \leq c' \leq C$ and $a' \in \mathcal{A}$.[1]

We say that a set of such hash functions *identifies* the set $\hat{A}$ if for every $a \in \hat{A}$ there is a $c : 1 \leq c \leq C$ such that for all $a' \in \hat{A}$ such that $a' \neq a$, it is true that $H_r^c(a') \neq H_r^c(a)$. We choose (α, C) so as to ensure that for any choice of $\hat{A} \subseteq \mathcal{A}$ of size up to S, the probability for $H_r^1, \ldots, H_r^C$ to identify $\hat{A}$ over a uniformly chosen r is at least 1/2. (For brevity, where this is not ambiguous we also say that "r identifies $\hat{A}$". If we mean that the property is true for a given $a \in \hat{A}$, we say that "r identifies a" or that "c identifies a".)

The full algorithm is as follows.

1. Let t be a tag held by the FIU, such that A is the set of t's keys.
2. The FIU sends to the RE the description of a set $\hat{A}$ that is a superset of A.
3. The RE determines $\hat{A}$, computes S and returns the value of S to the FIU.

[1] These are idealised assumptions. In practice, we rely on cryptographic strengths and statistical strengths of the hash functions to approximate these behaviours for practical purposes.

4. The FIU and the RE independently determine α and C.

5. The RE generates a random value r, computes $H_r^1, \ldots, H_r^C$ and determines whether these identify $\hat{A}$. If not, a new r is generated. The RE repeats this until an r is found that identifies $\hat{A}$.

6. The RE communicates the selected r to the FIU.

7. The FIU sends to the RE C vectors of semi-homomorphically encrypted values, $v_1, \ldots, v_C$, each vector of length S'. The content of each $v_i[j]$ is $\sum_{a \in A : H_r^i(a)=j} t[a]$. (Computing v_i values can be done efficiently, by looping over a and performing at each iteration $v_i[H_r^i(a)] \leftarrow v_i[H_r^i(a)] + t[a]$. However, it is better if the loop over a is padded with zero values to a total length of S if one wants to protect oneself from a potential non-data attack. For example, if the FIU responds to the RE too quickly, this will leak the information that A is small.) Values must be "refreshed" in the sense that these are all never-before-used ciphertexts. As a result, ciphertexts will also be all distinct.

8. Per assumption, for each $a \in \hat{A}$ there is a c_a that identifies a. The RE creates its tag, t', as having a key for every $a \in \hat{A}$, with $t'[a] = v_{c_a}[H_r^{c_a}(a)]$. If there is more than one such c_a, one can be chosen arbitrarily from the available options.

We leave it as an exercise for the reader to verify that the tag t' is identical to t, except for the addition of zero tag values to all $a \in \hat{A} \setminus A$.

Note 2.2. In a typical use scenario, the RE will only be interested in the zero/nonzero value of account tags, rather than their full numerical value. When this is the case, after the end of this algorithm (including its parts described in Section 2.2) and before any tag value is actually used, the RE should sanitise the tag (multiply each tag value by a random non-zero group element) to make sure any information in the tag other than its zero/non-zero status is destroyed.

In this document, for the purpose of computing communication sizes, we will assume that the v_i are sent with all of their ElGamal semihomomorphically-encrypted values stored in the Affine representation, this causing each tag value to be stored in 128 bytes.

In the typical case, where the FIU merely wants "1"s and "0"s as tag values, the values encrypted in the v_i will all be very small. If one simply stores in advance a cache of pre-encrypted, pre-compressed numbers, e.g. in the range from 0 to 20, that will cover all but an almost-negligible portion of the values required for the v_i, and one will almost never need to compute additional values on the fly. This solution allows us to halve the communication requirements with essentially no impact on running time.

In other cases, one can simply compute the desired values using summations in Projective representation, and perform the compression to the Affine representation (a division) only as a last step. This is still a much lighter operation than encryption. If more compression is needed, one can use the Folded representation for a further factor of 2 saving on communication costs, but doubling computation costs related to compression and de-compression.

Note that one is not even able to perform a "refresh" natively in Affine representation, which is why it is not enough to only maintain a cache of zeroes, as is done elsewhere in the FinTracer algorithm.

2.2 Honesty check

As will be demonstrated in Section 4, the process described so far is enough to provide protection against data leakage when all participants in the protocol are honest (but possibly curious) players.

We can add an additional step to achieve what is often called a *cheat-sensitive* protocol. This additional step, following the core algorithm, allows the RE to ascertain that the FIU has followed the protocol properly. The revised protocol assures the RE that with confidence $1 - \delta'$ the FIU has not deviated from the protocol, where δ' is a policy parameter of the algorithm.

In general, we advocate use of this honesty check. However, such assurance has its own security costs. See Section 6 for a discussion.

The validation process is as follows.

1. For each $a \in \hat{A}$ and $c \in [1, C]$, the RE subtracts $t'[a]$ from $v_c[H_r^c(a)]$. If the FIU has followed the protocol, the resulting v_c vectors should all be encrypted zeroes.

2. The RE sanitises the vectors and sums all values from all vectors together. Again, if the FIU has followed the protocol, the result, V, should be an encrypted zero.

3. The RE performs Zero-Knowledge validation that V is zero, using the protocol described in Section 2.3

Note 2.3. Previous versions of this document contained a description of a validation process that is susceptible to attack. See Appendix A for details.

2.3 Zero-Knowledge validation

We describe a protocol that allows an RE holding a value, V, that is encrypted using the FIU's key, to validate that V's plaintext value is (without loss of generality) zero. The intent is to use this protocol when $V = 0$ indicates that the FIU has followed the protocol and $V \neq 0$ indicates the opposite, deterministically. At the end of the protocol, both sides will discover the zero/non-zero status of V. If it is nonzero, all further processing should halt, and alerts should be raised both at the FIU and at the RE, because such an occurrence indicates deterministically that one of the parties has not followed the protocol.

The protocol uses a security policy parameter, δ'. It requires that if V is zero the protocol must always determine this, whereas if it is nonzero the protocol must determine this with probability at least $1 - \delta'$.

Note that while this is in the definition of the protocol requirements, it is nevertheless the case that determining whether V is zero or not is itself exploitable. See Sections 5.2 and 5.3 for a discussion.

Other than V's zero/nonzero status, however, provably no additional information is revealed to either side.

The way to ensure this is to use the Zero-Knowledge Proof protocol. We will describe this protocol here, providing the full specific details required for the present scenario.

To present the algorithm, recall that encrypted values are actually tuples, (a, b), such that $b = ax + m$, where x is the FIU's private key and m is the Ed25519-encoded version of the message. If a plaintext value is zero, m is zero, so $b = ax$. Otherwise, not. The protocol is as follows.

1. The RE sanitises V (if this was not done previously), then refreshes it. It then sends the result, $V' = (a, b)$, to the FIU.

2. The FIU uses its private key, x, to verify that $b = ax$, i.e. that V really was zero. If it is not, it aborts execution and alerts both sides.

3. Otherwise, the FIU generates a vector of $n = \lceil -\log_2 \delta' \rceil$ random, uniformly distributed, nonzero elements in $\mathrm{GF}(p)$, where p is the size of Ed25519. Let these be $\gamma_1, \ldots, \gamma_n$.

4. The FIU computes $b\gamma_1, \ldots, b\gamma_n$ and sends these to the RE. Let us name them $c_1, \ldots, c_n$.

5. The RE verifies that none of the c_i are zero. If any zeroes are in the list, it aborts execution and issues an alert to both sides.

6. Otherwise, the RE generates a vector of random, uniformly distributed Booleans, $q_1, \ldots, q_n$ and sends these to the FIU.

7. The FIU responds with a vector of GF(p) elements of length n, such that if q_i is "True", the corresponding response is $r_i = \gamma_i$. If it is "False", the response is $r'_i = x\gamma_i$.
8. The RE validates each response. If q_i is "True", the response is validated by checking that $c_i = br_i$. If it is "False", the response is validated by checking that $c_i = ar'_i$. If any of these checks fails, the RE aborts execution and issues an alert to both sides.
9. Otherwise, the RE declares that validation was successful.

The security of this algorithm is analysed in Section 4, where it is shown that the algorithm meets all required criteria, assuming the hardness of the discrete log problem (upon which the entire strength of ElGamal relies).

3 Choosing the parameters

We now show how to choose the various parameters utilised in the course of the algorithm, so as to meet desired privacy policy targets.

3.1 Choosing S

The first parameter to be chosen in the course of the algorithm is S. This should be an upper bound on $|\hat{A}|$, and preferably also an approximation of $|\hat{A}|$, but not so good an approximation as to provide an attacker with knowledge about whether any specific a is in $\hat{A}$ or not.

To meet this requirement, we will make the choice of S (ϵ, δ)-differentially private, where ϵ and δ are policy parameters of the algorithm.

For an algorithm $\mathcal{T}$ to be described as (ϵ, δ)-differentially private it must satisfy the following property. For any input X (described as a set), with probability at least $1 - \delta$ over the random choices of $\mathcal{T}$ it is the case that the output of the algorithm $x = \mathcal{T}(X)$ is such that

$$\text{Prob}[x = \mathcal{T}(Y)]e^{-\epsilon} \le \text{Prob}[x = \mathcal{T}(X)] \le \text{Prob}[x = \mathcal{T}(Y)]e^{\epsilon}$$

for every Y such that $|X \triangle Y| = 1$. Here, "$\triangle$" indicates symmetric set difference, "$|\cdot|$" indicates set size, and "e" is Euler's constant.

Let $AEXP(N, \gamma)$, for a nonnegative integer N and a positive γ, be the distribution over the integers in the range $[0, \infty)$ such that the probability that a random variable allotted from this distribution equals x is proportional to $e^{-\gamma|N-x|}$.

If the random variable $\mathbf{x} \sim AEXP(N, \epsilon)$ satisfies that

$$\text{Prob}[\mathbf{x} = 0] \le \delta, \tag{1}$$

then the randomised choice $S = |\hat{A}| + \mathbf{x}$ satisfies the desired condition of (ϵ, δ)-differential privacy.

In order to find an N that satisfies (1), consider that

$$\text{Prob}[\mathbf{x} = 0] = \frac{e^{-\epsilon N}}{\sum_{x=0}^{\infty} e^{-\epsilon|N-x|}} \le \frac{e^{-\epsilon N}}{\sum_{x=N}^{\infty} e^{-\epsilon|N-x|}} = e^{-\epsilon N}\left(1 - e^{-\epsilon}\right).$$

An N that fits our criteria is therefore

$$N = \max\left(0, \left\lceil \frac{1}{\epsilon} \log\left(\frac{1 - e^{-\epsilon}}{\delta}\right) \right\rceil\right),$$

with another possibility being the larger (but perhaps more elegant)

$$N = \max\left(0, \left\lceil \frac{\log \epsilon - \log \delta}{\epsilon} \right\rceil\right),$$

where here and throughout all logarithms are natural logarithms unless stated otherwise.

3.2 Choosing α and C

Per the design of the algorithm, we want to set α and C in such a way that a randomly chosen r will identify any set $\hat{A} \subseteq \mathcal{A}$ of size at most S with probability at least 1/2. This is a constraint on (α, C).

Consider, now, the communication costs of the algorithm. These are proportional to $C \times \alpha S$. So, for a chosen S, the costs are proportional to $L = \alpha C$. We will use L as a target function to be minimised.

To solve the problem, we begin by choosing α so as to minimise the probability p that r does not identify any specific account a in a set $\hat{A}$ of size S, given L. We then choose L such that, on the optimal α, the probability that $\hat{A}$ has not been identified is at most 1/2.

We begin, therefore, by determining α.

If we randomly, uniformly and independently map S values into $S' \approx \alpha S$ positions, the expected number to be mapped uniquely is approximately $S(1 - 1/(\alpha S))^{S-1} \approx S e^{-1/\alpha}$. This leaves $S(1 - e^{-1/\alpha})$. Therefore, $p \approx \left(1 - e^{-1/\alpha}\right)^C$. Over a constant $L = \alpha C$, this means α can be optimised by minimising $p \approx \left(1 - e^{-1/\alpha}\right)^{L/\alpha}$. The optimal α is $1/\log(2) \approx 1.44$.

This choice of α provides $p = 0.5^C$.

If we choose $C = 1 + \lceil \log_2 S \rceil$, this would ensure $p \leq \frac{1}{2S}$, indicating that the probability that r fails to identify *any* element of a set of size S is bounded by 1/2 (which is the sum of these probabilities) as desired.

To compute this numerically in an extreme case, let us assume we choose as $\hat{A}$ the set of all accounts, this being a set of size roughly 2^{25} for a single RE. The total communication size from the FIU to the RE in this case amounts to roughly 156GB, assuming communication in the Affine representation.

4 Security analysis

There are three types of potential security issues this algorithm needs to address.

1. The RE must not discover any new information.
2. An honest-but-curious FIU must not discover privacy-protected information on non-target accounts.
3. In the event of the protocol not being followed, the FIU must not discover privacy-protected information on non-target accounts.

We will consider these in turn.

4.1 Leakage to the RE

4.1.1 Core algorithm

The core algorithm meets its requirements of not leaking information to the RE trivially, by never communicating to the RE anything but a block of "fresh" ciphertexts in an amount that the RE has stipulated. Clearly, no information is leaked to RE in this process.

4.1.2 The honesty check

We will show that regardless of whether the RE follows the protocol or not, it cannot divulge any new information from the honesty check step.

The FIU receives from the RE a tuple (a, b). If both sides followed the protocol, this value satisfies $b = ax$, and the FIU immediately validates this or aborts. Attempts of the RE to use a tuple for which it does not know definitively that it satisfies $b = ax$ are a known and inevitable weakness of any honesty check. We discuss it in Section 5.3. For the purpose of this section, we will assume no such attempt is made, so $b = ax$ is satisfied.

Throughout the remainder of the protocol, the FIU merely sends to the RE the list of c_i values and its corresponding list of either r_i or r'_i values such that either $c_i = br_i$ or $c_i = ar'_i$, with both r_i and r'_i being uniformly distributed among GF(p)'s nonzero elements.

Note, however, that the RE could just as easily have allotted its own uniformly distributed value, be it r_i or r'_i, and could have just as easily computed br_i and ar'_i itself, with no help from the FIU. As such, it is not possible that the RE may have gained any new information from the protocol, other than the knowledge that the FIU, too, is able to calculate the desired values.

4.2 Honest-but-curious FIU

4.2.1 Core algorithm

Consider that the core algorithm consists of two sub-algorithms. First, there is the communication of S. This was already analysed in Section 3.1, where it was shown that S provides (ϵ, δ)-differential privacy.

Next, there is a communications round where the RE sends a challenge, r, being a random value containing no privacy-protected information, and the FIU responds with tag ciphertexts.

In this second round, it is not possible for the FIU to gain private information at all from the value of r because it is random. Even a non-data attack, attempting to derive information from the number of attempts to generate candidate values for r before the finding of a successful candidate, does not reveal much, because the total number of attempts is a geometric random variable with expectation roughly 2 regardless of $\hat{A}$.

We conclude, therefore, that there is no leak of RE private data towards the FIU in the core algorithm.

4.2.2 Honesty check

In the case where both parties are honest (but possibly curious), in the first communication round of the honesty check the RE merely sends a fresh zero, so does not convey to the FIU any new information.

In the second communication round, the RE transmits to the FIU a predetermined number of random Booleans, and thus again does not provide the FIU with any information.

4.3 Incorrect use of the protocol

4.3.1 Properties of the syndrome

An oblivious querying protocol raises the possibility of a mis-use of the protocol in order to cause a leak of information to the FIU about which accounts are held by an RE. We wish to gaurd against this potential for mis-use.

Consider the algorithm without the honesty checking step. If the FIU keeps all $v_i[j]$ as zero except for a single one, the resulting tag may either be all zeroes or have a single tag-positive account. Let us suppose an account with a nonzero tag value exists in the result t'.

Suppose (as an extreme example) that the FIU now asks the RE to report the identity of this account[2], then

1. The FIU has by this successfully "fished" the identity of a previously unknown account, and
2. The FIU has ascertained that $\hat{A}$ does not include any other accounts that would otherwise have been mapped to the tweaked $v_i[j]$ element.

To understand how the honesty check guards against such usage, it is useful to think of it as an error-correcting code. The encoding of the desired tag t is a linear transformation over t that replicates each t value into C separate positions in v. The RE-side calculation in the validation

[2]In [2] and [3], algorithms are discussed that support such queries.

process essentially computes the syndrome of v. This syndrome is zero if and only if v is a legal code-word, i.e. a vector that could have been derived by expanding a t.

If the syndrome is zero, the tag created by the RE, t', is identical to the original t, so the FIU would not gain new information even by querying t' directly.

Note that because each element a of A is mapped to C separate hash positions, even for a very small S this mapping contains enough information to uniquely identify a in all of $\mathcal{A}$. For example, even if S is of size 2^{10} (which is much too small to provide meaningful anonymity protection for the identity of A within $\hat{A}$), the hash positions of an account a provide $10 \times 11 = 110$ bits describing it. This is considerably more than the approximately 50 bits needed to uniquely identify a 15-digit string (or the approximately 40 bits needed to uniquely identify a 15-digit string beginning with a valid BSB).

Moreover, the dimensionality of the legal codewords is $|\hat{A}| \le S$, whereas the dimensionality of the space of all possible v values is $S'C \ge \alpha CS$. For this reason, it is not possible for the FIU to simply randomly assign values to hash positions in the hope of hitting against a legal codeword. The probability of such successful guessing is completely negligible.

4.3.2 Validation of the syndrome

The space of possible ciphertexts is the space of tuples (a, b) where a and b are elements in Ed25519. Let us partition it into two, as follows. Let $Z = \{(a, b)|b = ax\}$ and $N = \{(a, b)|b \ne ax\}$.

Because of the sanitisation and the refresh of V, if V is zero, V' is uniformly distributed in Z; otherwise it is uniformly distributed in N.

Thus, in the first communication round of the honesty check the RE does not convey to the FIU any information other than whether V is zero or not. If the FIU has followed the protocol, V will be zero and will this contain no information. If the FIU wishes to encode anything that contains information into V, this is likely to be detected.

In Section 5.2 we discuss in detail the possibility that the FIU may use the very risk of being detected as an information-delivery mechanism. In this section, however, we prove that if V is used to encode any nonzero value, this will be detected with high probability.

Specifically, in the previous section we have established that any attempt by the FIU to stray from the protocol by trying to "fish" for accounts inevitably lead to a nonzero syndrome. This, in turn, leads to a nonzero V, which leads to an (a, b) pair uniformly distributed in N. We now show that when (a, b) is thus distributed, this will necessarily be discovered by the honesty check with probability at least $1 - \delta'$.

To see this, consider that the value of b/a, x', is in this case uniformly distributed among all elements of the group except the FIU's private key x.

Being able to retrieve x' from (a, b) is therefore directly a solution to the "discrete log" problem, which is a problem whose hardness assumption underpins all of ElGamal encryption. We can therefore safely assume that the FIU does not, in this scenario, know the value of x'. We will show that being able to pass the honesty check with higher than $1 - \delta'$ probability is tantamount to knowing x'.

Specifically, consider that the FIU sends to the RE the value c_i (for $i = 1, \ldots, n$) and offers the RE to choose between discovering r_i such that $c_i = br_i$ and r'_i such that $c_i = ar'_i$. If the FIU does not know x', it cannot know both r_i and r'_i. Knowing both r_i and r'_i implies knowledge of x', because from r_i and r'_i one can easily compute x' as r'_i/r_i.

Thus, if $V \ne 0$, the FIU can know the answer to at most one of the two possible questions. By asking $n = \lceil -\log_2 \delta' \rceil$ such queries, and choosing randomly and independently in each case which answer to receive, the RE ensures that the probability of a departure from the protocol successfully evading detection is limited by $2^{-n} \le \delta'$.

5 Vulnerabilities

Despite the above analysis, it is still the case that the algorithm presented here has some vulnerabilities. These vulnerabilities are largely ones that are inevitable, and will exist in any solution algorithm to the same problem.

We enumerate these here for completeness. We note that none of them are relevant under the assumption that the parties are honest and follow the protocol correctly.

5.1 Tag tampering

First, it is important to note that just because the FIU was able to transmit a tag value to the RE, this provides no guarantee that this tag value will subsequently be used. Suppose, for example, that the RE wants to shield particular accounts from ever matching a typology, the RE can take the created tag t' and zero out the relevant values before using the tag. In practice, such a change would be undetectable.

This is not an information leak, however, and it is a problem relevant to all our algorithms. It is possible for REs to tamper with tags regardless of how they are generated.

5.2 FIU validation

An actual information leak can occur using a strategy of guess-and-validate. The FIU may guess that an account a is in $\hat{A}$, and can check this by using a t value that has a nonzero tag for a.

This is a risky proposition, because if a is not in $\hat{A}$ the attempt will trigger the algorithm's alerts. However, if the guess is correct it will not be detected, and the FIU will have by this gained information.

We refer to such a strategy as a "Guess correctly or get detected" (GD2) strategy.

Like the previous vulnerability discussed, this "flaw" is an inherent problem in any mechanism that allows the FIU to set tag values. The alternative (which is essentially equivalent to not performing the honesty check) is to *not* detect such departures from the protocol, but, as shown before, the cost of that is opening the RE to much more direct and practically feasible methods of fishing.

5.3 RE validation

An RE could also theorectically depart from the protocol and use a guess-and-validate scheme. The RE could, for example, guess that the tag value for a particular account, a, is zero, and based on this can replace the value of V with a's encrypted tag value when initiating the honesty check. If the guess is incorrect the attempt will be detected immediately, but if it is correct, the RE will have silently gained by this action knowledge.

As in the case of guessing by the FIU (discussed in Section 5.2), it is possible to avoid this potential problem simply by *not* performing the honesty check, and (as in the case of guessing by the FIU) doing so would open up feasible methods for an FIU to depart from the protocol to fish for RE accounts.

This tradeoff, too, appears to be inherent to any solution that performs / does not perform an honesty check on its input, regardless of the particular choice of algorithm. (For example, regardless of algorithm the RE may use its guess to effectively tamper with the FIU's input, so that if its guess is correct the tampering does nothing but otherwise it is detected.)

As an example of how an RE that departs from the protocol could utilise such a GD2 ("Guess correctly or get detected") strategy, the RE may guess regarding one or more accounts that they are *not* in A. This is easy to do by summing their tag values in t' and adding them to the value of $X[y]$. As long as there are no intel investigations directed at these accounts, such action will be undetectable. As soon as such investigations are initiated in the FinTracer system, this will trip up alarms. The RE's protocol departure will be detected, but the RE will have gained the information.

This, and the RE's ability to take an account off a tag in order to shield it (which was discussed in Section 5.1) are perhaps the "highest impact" dishonest actions that theoretically could be attempted, and their implications should be considered in any implementation.

6 Conclusions

We have presented an algorithm for obliviously setting the values of a tag, and have shown that any vulnerabilities of this algorithm are largely inevitable, and will appear in every competing algorithm as well.

More specifically, we presented two variants of this algorithm—one where there is no checking that the FIU is acting honestly and one where such a check exists—and have shown that they represent two families of solution algorithms, where the vulnerabilities in each are those inevitable for that entire family of solutions (solutions where the FIU's actions are validated vs solutions where they are not).

This being the case, the most salient point when considering whether such an algorithm is to be implemented, is which of the two variants presents the better tradeoff in the cases considered.

Table 1 summarises these tradeoffs. The left-most column presents the algorithmic vulnerabilities in the variant where the FIU's actions are not validated, the middle column—where they are validated, and the right-most column provides, for comparison, the situation in the non-oblivious case, where the FIU simply provides the tag list to the REs in plaintext. This last is an algorithm where an honest RE receives information that an oblivious algorithm would have prevented them from receiving, but in other dimensions it is comparable to an algorithm from the "validated" family. It is provided here in order to demonstrate that the remaining vulnerabilities are essentially independent of algorithmic choice, and are only about the choice of family.

	Non-validated	Validated	Non-oblivious
Honest FIU discovers	S	S	nothing
Honest RE discovers	$\hat{A}$	$\hat{A}$	the explicit A
DP RE can tamper	yes	yes	yes
DP RE	—	can GD2	Same as "honest"
DP FIU	can "fish"	can GD2	can GD2

Table 1: Comparison of vulnerabilities between the non-validated oblivious algorithm (left), the validated oblivious algorithm (middle), and a non-oblivious algorithm (right). "GD2" refers to the "Guess correctly or get detected" strategy. "DP" stands for Depart from Protocol

The last two lines of the table seem to be the most salient points to consider when choosing a strategy.

A Vulnerabilities of previous protocol

Previous versions of this document contained a validation protocol in which the RE validates that an encrypted value, V, equals zero by placing it in a set of nonzero values and asking the FIU to point out the zero. It was argued that the RE cannot gain by this information because it would have to generate a set of values where exactly one value is zero, so would need to have knowledge of the zero/non-zero status of each value, in any case.

This is, unfortunately, incorrect. The RE can generate a set containing exactly one zero even without knowing which of the elements in the set is the zero, and can use this in order to gain information.

For example, if the RE knows that for a particular account v a particular tag t has one of a set of possible values, $a_1, \ldots, a_n$, it can use, as part of the set sent to the FIU, the values $t[v] - a_1, \ldots, t[v] - a_n$. Exactly one of these is known to be zero, and if the FIU divulges to the RE which it is, the RE will have by this learned the true value of $t[v]$.

The new algorithm replacing this method is, by contrast, provably secure.

References

[1] Brand, M., Ivey-Law, H. (2020), FinTracer: A tracing mechanism for electronic money, AUSTRAC internal. (Available as: `fintracer.pdf`)

[2] Brand, M. (2022), The Complete, Authorised and Definitive FinTracer Compendium. (Available as: `FT_combined.pdf`)

[3] Brand, M. (2020), Improved differential privacy for FinTracer Boolean queries. (Available as: `boolean_queries.pdf`)