Secure Data Hiding for Contact Tracing

Craig Gotsman[1], Kai Hormann[2]

[1] New Jersey Institute of Technology, Newark, NJ, USA
[2] Università della Svizzera italiana, Lugano, Switzerland

Abstract

Contact tracing is an effective tool in controlling the spread of infectious diseases such as COVID-19. It involves digital monitoring and recording of physical proximity between people over time with a central and trusted authority, so that when one user reports infection, it is possible to identify all other users who have been in close proximity to that person during a relevant time period in the past and alert them. One way to achieve this involves recording on the server the locations, e.g. by reading and reporting the GPS coordinates of a smartphone, of all users over time. Despite its simplicity, privacy concerns have prevented widespread adoption of this method. Technology that would enable the "hiding" of data could go a long way towards alleviating privacy concerns and enable contact tracing at a very large scale. In this article we describe a general method to hide data. By hiding, we mean that instead of disclosing a data value x, we would disclose an "encoded" version of x, namely $E(x)$, where $E(x)$ is easy to compute but very difficult, from a computational point of view, to invert. We propose a general construction of such a function E and show that it guarantees perfect recall, namely, all individuals who have potentially been exposed to infection are alerted, at the price of an infinitesimal number of false alarms, namely, only a negligible number of individuals who have not actually been exposed will be wrongly informed that they have.

1. Introduction

Contact tracing has proven to be an effective tool in controlling the spread of infectious diseases such as COVID-19. It involves digital monitoring and recording of physical proximity between people over time, so that when one user reports infection, it is possible to identify all other users who have been in close proximity to that person during a relevant time period in the past and alert them. These users would be required to monitor their health and isolate, allowing early treatment and preventing further spread. Contact tracing has been deployed successfully in countries such as China, South Korea, Singapore, Israel, Australia, and Germany, and seems to be the only effective way to detect and contain the spread early in the process.

There are two main approaches to contact tracing. The first is based on the *relative* distance between users. Using the Bluetooth sensor on a smartphone, it is possible to detect signals from other users with Bluetooth emitters who are physically close by (i.e. within a certain range) and record the proximity, either locally on the user's device, or at a central authority/server. This method, currently under development by Apple, Google and others [1], has the advantage that

absolute locations of users are never disclosed, ensuring some degree of privacy. The disadvantage is the reliability of the Bluetooth sensors and their ability to work well under all relevant conditions (e.g. occlusion) and at all relevant ranges and some security concerns about the popular decentralized approach to storing this type of data on user devices.

The second approach to contact tracing involves recording on a central server the *absolute* locations, e.g. by reading and reporting the GPS coordinates of a smartphone, of all users over time. This obviously provides the server with more information to work with than the first approach, enabling not only alerts to nearby users, but also to identify geographic hotspots and other patterns of contagion. It also provides a historic record of the evolution of an epidemic which can be mined and analyzed in many other ways.

Despite the simplicity of the second approach, privacy concerns have prevented its widespread adoption. Many people do not want their location history to be known to any third party, thus would avoid using any software that explicitly discloses this information. Some have gone so far as to call contact tracing based on unprotected disclosure of location data illegal or unconstitutional [2]. A number of commercial contact tracing apps, which report and store explicit location data, have recently been found in violation of user privacy policies, having shared this data with unauthorized third parties [3]. Such privacy concerns must be addressed if automatic contact tracing is to be deployed, as it is not very effective unless adopted by a majority of the population.

Technology that would enable the "hiding" or "obfuscation" of location data could go a long way towards alleviating privacy concerns and enabling contact tracing at a very large scale. Since the outbreak of COVID-19, this has been the topic of recent research, incorporating cryptographic techniques such as *private set intersection* [4], *private proximity testing* based on an *equality testing protocol* [5] and *homomorphic encryption* [19]. We refer the interested reader to the comprehensive survey by Reichert et al. [20]. The objective of this article is to describe a very simple method to hide data, which can also be used to hide spatio-temporal data. By hiding, we mean that instead of disclosing a data value x, a user would disclose an "encoded" version of x, namely $E(x)$. For this to be useful, it should be easy for any user to compute $E(x)$ if given x, but be very difficult, from a computational point of view, to invert E, namely to recover x when provided only with $E(x)$ (even for the user who encoded x). By "difficult" we mean it would require a prohibitive amount of storage or of computational resources, which would effectively deter any such attempt. Although quite distinct, as we will make clear later, these resemble in spirit *one-way functions* or *cryptographic hash functions* used in classical cryptography. In its simplest form, the function E is deterministic and injective, as then it is easy to check if $x = y$ by simply checking if $E(x) = E(y)$. In the contact-tracing scenario, the data $x = (t, l)$ is a data value consisting of a concatenation of the time t with the location l. Given the function E, a user with ID i would periodically transmit to a central server the pair (i, e), where $e = E(x)$ is the encoded version of x. The server would store these pairs in a database indexed by the second component. Given a *query* vector e (of a detected infection), it should be easy to search this

database to determine all pairs (i', e') such that $e' = e$, namely identify which other users (having ID i') were also at location l at time t and alert them.

We depart from traditional cryptographic techniques by not requiring the use of encryption keys of any sort, neither private nor public keys. This means that even the user who computed $E(x)$ from x cannot recover x from $E(x)$ unless she explicitly records the connection between the two or stores some additional information which might facilitate the recovery. While the basic embodiment of E is deterministic, it is possible to add an extra layer of security by introducing a non-deterministic (probabilistic) element to E, namely $E(x)$ could assume more than one value for any given x. In this case we need to modify the database search to a *matching* procedure: given a query $e,$ instead of searching for other vectors e' such that $e = e'$, we search for all other vectors e' such that $\delta(e, e') \leq \tau,$ where δ is the Hamming distance function between two vectors, namely the number of coordinates in which they differ, and τ is some threshold. These e' are called *matches* of e. An exact match is, of course, the special case where $\tau = 0.$ A judicious choice of the encoding function E and the value τ will guarantee no false negatives (i.e. perfect *recall*), namely, given a query e corresponding to some data x, we will always find *all* other matching values e' corresponding to the same x. It will also guarantee a negligible (ideally zero) number of false positives (also called *false alarms*), namely, almost never report values e' corresponding to a different data value $y \neq x$. In the contact tracing scenario, perfect recall is necessary so that *all* individuals who have potentially been exposed to infection *are alerted*. A tiny number of false positives are tolerable as all this means is that a small number of individuals who have not actually been exposed will be informed that they have.

This article proposes encoding functions for spatio-temporal data. In a nutshell, it maps a 2D location l and time t, combined and represented as a large integer in a discrete world, to an n-dimensional vector of integers $E(x)$, where n is quite large, e.g. 100. The range of the components of $E(x)$ can be much larger than n, e.g. $\{0, \ldots, 502\}$. The function E is based on well-known number-theoretic techniques, the preferred one making use of polynomials over finite fields. First deployed in 1960 in Reed–Solomon error-correcting codes [6] and its variants (the most important being the BCH code), the technique has also found use in other cryptographic methods, such as Shamir's secret sharing method [7] and even blockchain [8]. The most important property of E is that it transforms a very large integer into a long vector of much smaller integers in an injective way, which can be thought of as an *embedding* in a higher-dimensional space, and this transformation cannot be inverted *unless a minimal number $m \leq n$ of the vector coordinates (and their indices in the vector) are known.* We take advantage of this by *sorting* the vector coordinates so that their correspondence to the coordinate indices is lost, making it difficult to apply the standard decoding methods. An attacker has no choice but to try all possible permutations of subsets of size m of the n coordinates, making it computationally infeasible, even for relatively small values of n and m. Another important property is that, although there are simple algebraic relationships between the coordinates of the vector, to the naked eye, and even to a statistical test, they look like random integers. Thus, the distribution of the encoded vectors in the embedding space is quite uniform, which will work in our favor.

2. The Setup

Consider an integer domain (the "world"): $W = \{0, \dots, M-1\}$. Any integer $x \in W$ is a valid message and we may express it as a sequence of m digits $x = (x_1, \dots, x_m)$ in base p: $x = \sum_{i=0}^{m-1} x_i p^i$ where p is a prime number (or more generally a prime power) and $x_i \in \mathbb{Z}_p = \{0, \dots, p-1\}$. Note that this implies that $m = \lceil \log_p M \rceil$ and taking a larger m is superfluous. Essentially, W is synonymous with a subset of $\mathbb{Z}_p^m$, the set of all vectors of length m, where each coordinate is taken from $\mathbb{Z}_p$.

In the contact tracing application, the spatio-temporal world consists of two-dimensional (latitude and longitude) GPS coordinates at 1 meter resolution (or the Open Location "Plus" Codes [9]), which translates to a grid with 10^{14} points, and 10^5 different time stamps for every 30 seconds over the past month, implying a "world" of size $M = 10^{19}$. If we use the prime $p = 503$, this would mean $m = 8$.

3. The Encoding Function

We propose the following non-deterministic encoding scheme:

Let $W = \{0, \dots, M-1\}$ be an integer domain, n a positive integer and p a prime. Denote by $\mathbb{Z}_p^n$ the set of vectors with n elements from $\mathbb{Z}_p$ and by Δ_p^n the set of vectors with n elements from $\mathbb{Z}_p$ in non-decreasing order, also known as the *ordered discrete simplex*. The encoding function $E: W \to \mathcal{E} \subset \Delta_p^n$ has parameters (M, p, n, k), where $0 \le k \le n \le p$ and $n \ge m = \lceil \log_p M \rceil$.

To compute $E(x)$ for a domain element $x \in W$:

1. Express x in base p: $x = \sum_{i=0}^{m-1} x_i p^i$.
2. Compute the *basic* encoding $C(x) = (\pi(0), \pi(1), \dots, \pi(n-1)) \in \mathcal{C} \subset \mathbb{Z}_p^n$, where $\pi(\xi) = \sum_{i=0}^{m-1} x_i \xi^i \pmod{p}$ is a polynomial of degree $m-1$ over the finite field $\mathbb{Z}_p$.
3. Sort the coordinates of $C(x)$ in non-decreasing order to $C'(x) \in \mathcal{E}$.
4. Randomly modify k arbitrary coordinates of $C'(x)$, while preserving the increasing order of the coordinates, resulting in $E(x) \in \mathcal{E}$.

Note that as a result of step 4, $k > 0$ implies that $E(x)$ is non-deterministic, namely may assume multiple values.

The basic code *space* $\mathcal{C} \subset \mathbb{Z}_p^n$, defined as the set of all possible basic codes of world elements $\mathcal{C} = \{C(x): x \in W\}$ consists of vectors of length n, such that $C_i(x) \in \mathbb{Z}_p$. It has the following properties:

1. $C(x)$ is injective, namely $x = y$ iff $C(x) = C(y)$.
2. $\mathcal{C}$ has *Hamming distance* $d = n - m + 1$, namely any two distinct codewords $c_1, c_2 \in \mathcal{C}$ differ from each other by at least d coordinates: $\delta(c_1, c_2) \ge d$. This is because any

polynomial of degree $m - 1$ over a field is uniquely determined by m of its values. So not only is C an injective function (i.e. $d > 0$), but it maps distinct world elements quite far apart from each other in C.

3. x may be recovered from $C(x)$ by a variety of efficient methods, including inverting a linear Vandermonde system.

The basic coding function C described above was proposed by Reed and Solomon [6] as an error-correcting code to overcome corruption of $k = \lfloor d/2 \rfloor$ coordinates of $C(x)$. When presented with c', which is a corrupted version of $C(x)$, Property 2 guarantees that $C(x)$ is the unique codeword in C such that $\delta(C(x), c') \le k$, thus error-correction performed by replacing c' with the vector closest to it in C by the Hamming distance, is well-defined and yields the correct result $C(x)$. The corrected codeword $C(x)$ may be found by efficient algorithms (e.g. [10]) which take into account the special algebraic structure of C.

Our non-deterministic encoding function is a variation on the theme of error-correction. In our scenario, we are presented with *two* vectors $E(x)$, $E(y) \in \mathcal{E}$ originating from $x, y \in W$. We would like to have a threshold τ such that $x = y$ iff $\delta(E(x), E(y)) \le \tau$.

To give the flavor of our approach, we remark that it is relatively easy to determine this threshold if the encoding procedure does *not* contain the sorting step 2 in the encoding procedure, as the following lemma implies.

Lemma 1. If we eliminate the sorting step 2 in the encoding procedure with parameters (M, p, n, k), and set $k = \left\lfloor \frac{n-m}{4} \right\rfloor$ and $\tau = 2k$ for $m = \lceil \log_p M \rceil$, then $x = y$ iff $\delta(E(x), E(y)) \le \tau$.

Proof. From the definition of k, we have $n - m \ge 4k$, so
$$x = y \implies C(x) = C(y) \implies \delta(C(x), C(y)) = 0 \implies \delta(E(x), E(y)) \le 2k = \tau$$
$$x \ne y \implies \delta(C(x), C(y)) \ge n - m + 1 \implies \delta(E(x), E(y)) \ge n - m + 1 - 2k > 4k - 2k = \tau$$
$\square$

While not incorporating the sorting step 2 is amenable to easy analysis and identification of k and τ, it also compromises the security of the encoding $E(x)$, namely, it is then quite easy to recover x from $E(x)$. This is essentially error-correction from k errors, which, as mentioned above, is possible by a number of efficient algorithms, taking advantage of the special algebraic structure of C [10].

The advantage of introducing sorting step 2 is precisely because it prevents the use of the standard error-correction algorithms, since the critical correspondence between the coordinates of $C'(x)$ (and thus of $E(x)$) and the indices in the original $C(x)$ is lost.

The disadvantage of introducing sorting step 2 is that it modifies the Hamming distance d present in C, which is not likely to be preserved in C' and $\mathcal{E}$. In theory it could increase the distance, but it is much more likely to decrease it. It seems like it will be difficult to obtain a lower bound on this distance (which could have then been used to determine a threshold τ, akin

to Lemma 1), since all the algebraic structure that was present in C has been destroyed in the transition to C' and $\mathcal{E}$.

Luckily, we are still able to make useful observations about the nature of the encoded vectors in $\mathcal{E}$. To the naked eye, the basic code space C will consist of integer vectors of essentially random values in the range $\{0, \dots, p-1\}$. By "random" we mean actually pseudo-random, namely that although completely determined by x, it will be statistically impossible to distinguish between these vectors and completely random vectors. The sorting of the vectors will make them less random, but it will still be quite difficult to distinguish between the vectors in $\mathcal{E}$ and random non-decreasing integer vectors.

4. The Matching Algorithm

Let us recall the application: We have a database of D pairs of user ID's and encoded spatio-temporal values: $\{(i, E(x)) : i = 1, \dots, D\}$. Given the *query* – a vector e – we wish to find all *matches* of e, namely, find all database entries $\{(i, e')\}$ such that both e and e' are possible encodings of the same data value x, i.e. $\delta(e, e') \leq \tau$ for a suitable threshold τ. We say that τ is the *matching threshold* and e' *matches* e.

Recall that the size of the world is $M = |W| = 10^{19}$. Assuming 1 billion = 10^9 users, each storing location data for every 30 seconds over the past month, namely, close to 10^5 time-stamped locations, this implies that the database could contain $D = 10^{14}$ entries.

We would like to show that even though the vectors are sorted, a matching threshold of $\tau = 2k$ for "reasonable" values of k, as in Lemma 1 above, is still a good choice. This is because the size of the database (D) is much smaller than the size of the world (M), thus the probability that database vectors match a typical query vector is infinitesimally small, unless they are encodings of the same world data.

Remember that $D \ll M \ll N$, where $M \approx p^m$ and $N = p^n$. Now, if given a query $e = E(x)$ for which there exists a matching database entry e', then obviously $\delta(e, e') \leq 2k$. So to avoid false negatives, namely, to avoid missing correct matches, we must take $\tau \geq 2k$.

Can we expect a given query vector $e = E(x)$ to "accidentally" match a vector $e' = E(y)$ corresponding to another $y \neq x$ in the database because of the sorting and corruption of the original basic code vectors in C ? The following theorem implies that this false positive is highly unlikely.

Theorem 1: Given any $e \in \mathcal{E}$, an upper bound for the probability of a vector $e' \in \mathcal{E}$, generated by sorting the coordinates of a random vector $z \in \mathbb{Z}_p^n$, differing from e in at most τ non-adjacent coordinates is

$$\text{Prob}\{\delta(e, e') \leq \tau\} \leq s(p, n, \tau) = \frac{n!}{p^n} \cdot \sum_{d=0}^{\tau} \frac{(2p)^d}{d!}.$$

Proof. For the case $\tau = 0$, the probability of an exact match in all coordinates is at most $n!/p^n$, since all $n!$ permutations of e can be taken as z among all p^n possible unsorted vectors in $\mathbb{Z}_p^n$,

such that $\delta(e, e') = 0$. For every coordinate of e that occurs with multiplicity $\mu > 1$, the probability reduces by a factor of $\mu!$, because the order of the repeated coordinate in z does not matter.

For the case $\tau = 1$, let us study the number of sorted vectors $e' \in \mathcal{E}$ that differ from e in exactly one coordinate. Letting $e_0 = 0$ and $e_{n+1} = p - 1$, it is clear that each coordinate e_i' of e' for $i = 1, \dots, n$ can take any value in $\{e_{i-1}, \dots, e_i - 1, e_i + 1, \dots, e_{i+1}\}$ without compromising the correct order. Hence, there are

$$\sum_{i=1}^{n}(e_{i+1} - e_{i-1}) = p - 1 + e_n - e_1 \le 2p - 2 \le 2p$$

sorted vectors $e' \in \mathcal{E}$ at distance $\delta(e, e') = 1$ from e and thus the number of sorted vectors $e' \in \mathcal{E}$ with $\delta(e, e') \le 1$ is at most $2p + 1$. Using the same permutation argument as before, this proves the upper bound for $\tau = 1$.

For the case $\tau > 1$ we apply the previous argument iteratively τ times while using the assumption that the coordinates of e' that differ from those of e are non-adjacent. Then a vector at distance $\tau + 1$ is just a modification of a vector at distance τ in one additional coordinate, thus the number of modifications is at most $(2p)^\tau$. Note that this is an overestimate as a modification may occasionally *reduce* the distance by one. Since the order of modification of the modified coordinates is not important, we have counted each distinct modification $\tau!$ times. □

The assumption that the differing coordinates of e and e' are non-adjacent makes the proof of Theorem 1 easier, but we have experimentally observed that this upper bound holds also for the unrestricted case.

So the expected number of false positives for any given query e is at most $Ds(p, n, \tau)$, which decreases as τ decreases. For the values $p = 503$, $n = 100$, we may use $k = 10$ and matching threshold $\tau = 20$, thus $s(p, n, \tau) \approx 10^{-71}$. Since $D = 10^{14}$, the expected number of false positives per query is infinitesimal (10^{-57}), and even the expected number of false positives when each database entry is used as a query is still only $D^2 s(p, n, \tau) = 10^{-43}$.

Conclusion. In our encoding scheme, it suffices to take a corruption parameter k which is not too small and not too large, and then use $\tau = 2k$ as the matching threshold. Such a threshold will completely avoid false negatives and produce a negligible number of false positives.

Retrieving Matching Data. Now that we have a suitable matching threshold for our matching algorithm, we must address the algorithmic question of how to organize the database of D encoded values (which are sorted integer vectors), such that given a query vector e, it is possible to efficiently find all pairs (i', e') in the database such that e' matches e, namely such that $\delta(e, e') \le \tau$? This is known as the "static Hamming distance range query". Of course, exhaustive search of the database is possible, but that would cost $O(D)$ time, which is too costly in our scenario where $D = 10^{14}$. Efficient data structures have been devised for dealing with this

problem, as in Manku et al. [11]. This requires $O(\tau n D)$ storage (which is significant but not prohibitive in our application) but has very fast ($O(\log D)$) query runtime. See also Liu et al. [12] for more recent work.

5. The Tracing Algorithm

Now that we have an encoding algorithm and are able to match two encoded vectors, we describe the procedure to be followed by the individual users and the central server to do the actual contact tracing and alerts.

User with ID i

- The user continuously transmits to the server data pairs (i, e) where $e = E(x)$ and $x = (t, l)$ is her time and location, tagged as "uninfected". The user also stores the triples (t, l, e) in a local database indexed by t and e (e.g. on her smartphone), so that it is easy to retrieve all e's transmitted during a given time interval and recover (t, l) from its encoding e.

- If the user discovers she is infected, she sends *again* all pairs (i, e) generated by her over the past, say, two weeks (by querying her local database) back to the server, tagged as "infected".

- Upon receipt of message e tagged with "possible infection" from the server, the user recovers the infection time and location (t, l) from e (by querying her local database). The user self-isolates for two weeks and can possibly report (t, l) separately to friends and family.

Central server

- Upon receipt of a data pair (i, e) tagged "uninfected", the server stores the pair on the server database (of size D).

- Upon receipt of a pair (i, e) tagged "infected", the server retrieves from the server database (by the matching algorithm described in Section 4) all pairs (i', e') for which e' matches e. The server then sends these e' to user i' tagged with "possible infection".

6. Attacking the Code

Recall that a critical objective is to "hide" the data by its encoding, namely render it computationally infeasible to recover the (large) integer $y \in W$ from the integer vector $e = E(y) \in \mathcal{E}$, either because it would require too much computation time or too much storage space. We describe here three possible methods of attack and argue that they are infeasible.

Brute-force attack. The simplest method is just to exhaustively scan the entire world and check if the encoded version $e' = E(x)$ of any world point x matches the given encoding e (namely, that $\delta(e, e') \le \tau$). This would require $|W| = M = 10^{19}$ encodings and comparisons, which is prohibitive in runtime.

Table attack. We could reduce the runtime of the brute-force attack by trading off space for time, employing a very large database. Simply compute some encoding $E(x)$ for every possible

$x \in W$ in a preprocessing phase and store the pairs $(x, E(x))$ in a database indexed by $E(x)$. Given an encoding e, the matching algorithm described in Section 4 would then be able to quickly retrieve all matches of e. However, this requires a database of size $O(\tau n M)$ which is $\frac{M}{D} = 10^5$ times larger than the server database. For $M = 10^{19}, p = 503, n = 100$ and $\tau = 20$, this is at least 10^{21} bytes, and would be prohibitively large.

Direct attack. A direct attack occurs when an adversary tries to invert the encoding through a subset of the coordinates by applying the traditional decoding algorithms such as solving a linear Vandermonde system. This is foiled by the sorting of the coordinates of the vectors. Since inversion requires knowledge of the correspondence between coordinates and their indices for at least m uncorrupted coordinates, this is what an attempt to invert $e = E(x)$ must look like:

for each subset of m coordinates (out of n)   // there are $\binom{n}{m}$ such subsets
    for each permutation of m indices (out of n)   // there are $\frac{n!}{(n-m)!}$ such permutations
      solve for x (e.g., by multiplying e by the inverse of the Vandermonde sub-matrix consisting of the corresponding m rows from the full $n \times m$ Vandermonde matrix);
      if $x \geq M$ then continue;
      compute $e' = E(x)$;
      if $\delta(e, e') \leq \tau$ then return(x);
    end for;
end for;

Each solve costs $\Omega(m)$ time. Should any of the selected subset of m coordinates be corrupted, the inner loop will run completely, costing $\frac{n!}{(n-m)!}$ solves. Since the probability that none of the m coordinates are corrupted is $\left(1 - \frac{k}{n}\right)^m \approx \exp\left(-\frac{km}{n}\right)$, the outer loop will terminate on the average after $\exp\left(\frac{km}{n}\right)$ iterations and the inner loop will compute an expected number of $\frac{n!}{2(n-m)!}$ solves the last time it runs. Note that failure in one iteration due to one or more corrupted coordinates will not reveal which of the m coordinates are corrupted, so that there is no extra information that can help to choose a "better" set of m coordinates in the next iteration. In total, the expected number of solves for this attack would be $\frac{n!}{(n-m)!}\exp\left(\frac{km}{n}\right)$. For $n = 100$ and $p = 503$, we have $m = 8$. With $k = 10$, the expected number of solves is 10^{16}, which would take too long.

7. An Alternative: Redundant Residue Number Systems

While we have presented an encoding method based on polynomials over finite fields, it is possible to use another method which is also employed in error-correcting coding and secret-

sharing. This involves so-called *redundant residue number systems*. Originally proposed in the 1950's for efficient arithmetic computations on large integers [13], this technique was adopted for error-correction coding soon after [14,15] and is also used in cryptography [16,17]. The main difference between this method and the basic coding method described above based on polynomials is that now the basic code space is $C = \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_n}$ instead of $\mathbb{Z}_p^n$ for a sequence of distinct primes $(p_1, \ldots, p_n)$.

Recall that the "world" is $W = \{0, \ldots, M-1\}$. Let $(p_1, \ldots, p_n)$ be a sequence of increasing primes and m an integer such that $\prod_{i=n-m+2}^{n} p_i < M < \prod_{i=1}^{m} p_i$, Denote $N = \prod_{i=1}^{n} p_i$. The encoding function $E: W \to \mathcal{E}$ for a domain element $x \in W$, has parameters $(p_1, \ldots, p_n, k, n)$, where p_i are primes and $0 \le k \le n$ is an integer. The basic coding function is simply $C(x) = (x \,(\text{mod } p_1), \ldots, x \,(\text{mod } p_n)) \in C$. Similar to the case of polynomials over finite fields, the infamous Chinese Remainder Theorem [15] guarantees that x can be recovered from any subset of m coordinates of $C(x)$ along with their indices, so this code also has Hamming distance $n - m + 1$, and error-correction may be done using a variety of methods taking advantage of the algebraic structure (e.g. [18]). Our encoding proceeds as above, by sorting the coordinates of the basic code and corrupting a small subset without changing the order. Nothing else is changed.

Despite this approach actually being simpler to implement than the polynomial-based approach, it is less desirable due to m being more constrained as a function of the primes used. For example, for $n = 80$, taking p_i to be all the consecutive primes from 877 to 1,451 (having geometric mean 1,143) yields only $m = 7$. An appropriate k would be 8, thus $\tau = 16$. The probability of a false positive is then 10^{-58} and the complexity of the direct attack is 10^{13} (see Table 1).

Table 1. Parameters of the different settings for world size $M = 10^{19}$ and database entries $D = 10^{14}$.

coding method	encoded vector length	alphabet size	data vector size (base p)	corrupted coordinates	matching threshold	encoded vector size (bits)	expected # of false positives	"direct attack" complexity
	n	p	$m = \lceil \log_p M \rceil$	k	τ	$\lceil n \log_2 p \rceil$	$D^2 s(n, p, \tau)$	$\frac{n!}{(n-m)!} \exp\left(\frac{km}{n}\right)$
polynomial	100	503	8	10	20	898	10^{-43}	10^{16}
polynomial	100	101	10	1	2	666	10^{-10}	10^{20}
polynomial	200	211	9	20	40	1,545	10^{-5}	10^{21}
residues	80	$\approx 1,143$	7	8	16	858	10^{-58}	10^{13}

8. Discussion

Increasing the security. It is relatively easy to increase the security of the system, i.e. making a direct attack on the system more difficult. In the scenario described above, where $M = 10^{19}$, we took $n = 100$, $p = 503$, implying $m = 8$, thus the complexity of a direct attack is 10^{16}. If we were to take instead $n = 100$ and $p = 101$, so that $m = \lceil \log_{101} 10^{19} \rceil = 10$, the complexity would increase to 10^{20} (although we would have to take $k = 1$ and $\tau = 2$ to keep the probability

of a false positive at 10^{-10}), and if this were not enough, we can increase this further by increasing both n and p. See Table 1 for a comparison of the attack complexity resulting from different values of the system parameters. Increasing n obviously increases the (bit) size $n \cdot \log_2 p$ of the code $C(x)$ and thus the size of the server database, but the same is true for the database of the "table" attack.

Using a deterministic mapping. Our encoding method is non-deterministic, namely involves randomly corrupting a subset of $k > 0$ coordinates in the sorted basic code vector. The advantage of a large k is that it increases the difficulty of a direct attack on the database, as described in Section 6. However, for certain values of the other system parameters, it may be possible to make do with a deterministic encoding method, namely $k = \tau = 0$. In this case, matching a query vector within the server database reduces to exact vector match, which may be done easily by binary search on a table (of size D) of the database entries (i, e), sorted in lexicographic order of e.

Detecting proximity. The method outlined in this article provides an easy way to determine whether $x = y$ by comparing $E(x)$ and $E(y)$. Recall that x and y are taken from a discrete world, which are essentially samples of the true continuous world at some finite resolution grid. However, sometimes in contact tracing it is necessary to also determine proximity beyond the grid resolution, either because of an increased radius of infection or simply because the accuracy of the measured location (typically taken from a GPS device) is much worse than the grid resolution and the chances of an exact match in measured location even when two users are within grid resolution, is very slim.

It would seem difficult to achieve this, since the encoded vectors have a pseudo-random distribution and any spatio-temporal correlation between two data points would be "lost in encoding". The easy way to circumvent this is for the user to transmit to the central server encodings of not just her current location, but also of the neighboring grid points, effectively "dilating" the data point. This would incur some overhead in storage and transmission costs on both client-side and server-side.

"Inflating" the world. The world size, in our contact tracing application, is $M = 10^{19}$ integers, which is very large, but constrains some of the parameters in our encoding scheme. In particular, the parameter m, if too small, could compromise the security against the direct attack, as described in Section 6. One way to rectify this would be to "inflate" the world by means of some function $f : W \rightarrow W'$ with $M = |W| \ll |W'| = M'$. This function f should be injective and non-polynomial, so that it cannot be inverted easily at each individual coordinate. One possibility for such an f is the following:

Let q_i denote the i-th prime (i.e., $q_1 = 2, q_2 = 3$, etc.) and observe that the product of the first $m_0 = 16$ primes is a little larger than the size of our world. Hence, the first step is to map $x \in W$ to the residue code vector w.r.t. these 16 primes, namely compute $C(x) = (c_1, \dots, c_{m_0})$ with $c_i = $

$x \pmod{q_i}$. For the next step, let $s_i = \sum_{j=1}^{i-1} q_i$ denote the sum of the first $i - 1$ primes (i.e., $s_1 = 0, s_2 = 2, s_3 = 5$, etc.) and let us map each c_i to the $(s_i + c_i + 1)$-th prime, giving the vector $C'(x) = (c_1', \dots, c_{m_0}')$ with $c_i' = q_{s_i + c_i + 1}$. Finally, we define $f(x) = \prod_{i=1}^{m_0} c_i'$ and note that $f(x)$ is a *square-free* integer with exactly m_0 prime factors. Moreover, as the mapping C is injective, it follows that $f(x)$ and $f(y)$ for $x \neq y$ have at most $m_0 - 1$ common factors, thus guaranteeing the injectivity of f. The size of the inflated world is $M' = \prod_{i=1}^{m_0} q_{s_{i+1}} \approx 10^{39}$. We now continue to encode $x' = f(x) \in W'$ instead of $x \in W$ with the polynomial-based approach outlined above, but now having the advantage of a larger $m' = 15$ instead of the previous $m = 8$.

Other linear codes. The basic code based on polynomials that we use is a linear code, in the sense that the coding operation is just multiplication by a matrix: $C(x) = V \cdot x$ over $\mathbb{Z}_p$. V is the $n \times m$ Vandermonde matrix, which has the special property that *all* submatrices of size $m \times m$ have full rank. This property allows to recover x from *any* subset of m coordinates of $C(x)$ by multiplying them by the inverse of the appropriate submatrix of V. Thus *any* $n \times m$ matrix with similar properties would serve the same purpose. Furthermore, were we to construct an $n \times m$ matrix A with the property that *some* of the submatrices of size $m \times m$ have rank less than m, and that full rank is obtainable only when the submatrix is enlarged to $(m + l) \times m$, this, coupled with the corruption of coordinates during encoding, could further complicate the direct attack on the method described in Section 6.

References

[1] https://www.apple.com/covid19/contacttracing [accessed on 07/07/2020]
[2] https://www.techrepublic.com/article/data-researchers-at-odds-will-americans-opt-in-or-out-of-covid-19-contact-tracing-apps/ [accessed on 07/07/2020]
[3] https://www.msn.com/en-us/news/technology/researchers-found-north-dakotas-contact-tracing-app-covertly-sending-location-and-advertising-data-to-third-parties/ar-BB14sOCn [accessed on 07/07/2020]
[4] A. Berke, M. Bakker, P. Vepakomma, K. Larson, A. Pentland. Assessing disease exposure risk with location data: A proposal for cryptographic preservation of privacy. arXiv:2003.14412 (8 April 2020).
[5] J. K. Fitzsimons, A. Mantri, R. Pisarczyk, T. Rainforth, Z. Zhao. A note on blind contact tracing at scale with appli-cations to the COVID-19 pandemic. arXiv:2004.05116 (10 April 2020).
[6] I. S. Reed, G. Solomon. Polynomial codes over certain finite fields. J. SIAM **8**(2), 300–304 (1960).
[7] A. Shamir. How to share a secret. Commun. ACM, **22**(11), 612–613 (1979).
[8] L. Cheng, J. Liu, C. Su, K. Liang, G. Xu, W. Wang. Polynomial-based modifiable blockchain structure for removing fraud transactions, Future Generation Computer Systems **99**, 154–163 (2019).

[9] https://plus.codes/ [accessed on 07/07/2020]

[10] S. Gao. "A new algorithm for decoding Reed–Solomon codes" in Communications, Information and Network Security, V. K. Bhargava, H. V. Poor, V. Tarokh, S. Yoon, Eds. SECS 712 (Springer, 2003), pp. 55–68.

[11] G. S. Manku, A. Jain, A. D. Sarma, "Detecting near duplicates for web crawling" in Proc. 16th WWW (2007).

[12] A. X. Liu, K. Shen, E. Torng, "Large scale Hamming distance query processing" in Proc. IEEE Conf. Data Engineering, pp. 553–564 (2011).

[13] H. L. Garner, "The residue number system" in Proc. ACM Western Joint Computer Conf., pp. 146–153 (1959).

[14] R. W. Watson, C. W. Hastings, "Self-checked computation using residue arithmetic" in Proc. IEEE **54**, 1920-1931 (1966).

[15] F. Barsi, P. Maestrini, Error correcting properties of redundant residue number systems. IEEE Trans. Comp. **C-22**(3), 307–315 (1973).

[16] M. Mignotte, "How to share a secret" in Cryptography. EUROCRYPT 1982, T. Beth, Eds. LNCS 149 (Springer, 1983), pp. 371–375.

[17] C. A. Asmuth, J. Bloom, A modular approach to key safeguarding. IEEE Trans. Inf. Theory, **IT-29**(2), 208-210 (1983).

[18] O. Goldreich, D. Ron, M. Sudan, Chinese remaindering with errors. IEEE Trans. Inf. Theory **46**(4), 1330–1338 (2000).

[19] J. Bell, D. Butler, C. Hicks, J. Crowcroft. TraceSecure: Towards privacy preserving contact tracing. arXiv:2004.04059 (8 April 2020).

[20] L. Reichert, S. Brack, B. Scheuermann. A survey of automatic contact tracing approaches. Eprint:2020.672 (19 June 2020).