Locality-Sensitive Hashing Does Not Guarantee Privacy! Attacks on Google's FLoC and the MinHash Hierarchy System

Florian Turati
ETH Zurich
Zurich, Switzerland
florian.turati@inf.ethz.ch

Karel Kubicek
ETH Zurich
Zurich, Switzerland
karel.kubicek@inf.ethz.ch

Carlos Cotrini
ETH Zurich
Zurich, Switzerland
ccarlos@inf.ethz.ch

David Basin
ETH Zurich
Zurich, Switzerland
basin@inf.ethz.ch

ABSTRACT

Recently proposed systems aim at achieving privacy using locality-sensitive hashing. We show how these approaches fail by presenting attacks against two such systems: Google's FLoC proposal for privacy-preserving targeted advertising and the MinHash Hierarchy, a system for processing location trajectories in a privacy-preserving way. Our attacks refute the pre-image resistance, anonymity, and privacy guarantees claimed for these systems.

In the case of FLoC, we show how to deanonymize users using Sybil attacks and to reconstruct 10% or more of the browsing history for 30% of its users using Generative Adversarial Networks. We achieve this only analyzing the hashes used by FLoC. For MinHash, we precisely identify the location trajectory of a subset of individuals and, on average, we can limit users' trajectory to just 10% of the possible geographic area, again using just the hashes. In addition, we refute their differential privacy claims.

KEYWORDS

LSH, FLoC, MinHash, SimHash, Privacy

1 INTRODUCTION

Locality-sensitive hashing (LSH) [26] is a group of hash functions that map, with high probability, similar objects to the same hash. Comparing hashes instead of entire objects then results in an efficient procedure that has been used, for example, for plagiarism detection [32], detecting duplicate websites or images [17, 22], dimensionality reduction [5], and clustering [15].

Recent works [2, 9, 21, 27] have used LSH to process sensitive data, where it is assumed that the hashes can be made public without compromising the users' privacy. For example, Google proposed FLoC [27], a method for private targeted advertising. FLoC uses LSH to map browsing histories to hashes such that users with similar browsing histories likely have the same hash. The hashes are then grouped into *cohorts*. The idea is that each cohort contains users with similar browsing histories. The advertiser then learns only each cohort's identifier rather than each user's browsing history.

A second example is Apple CSAM [2], designed to detect child sexual abuse material in iCloud photos while also preserving user privacy. It uses LSH to map images to hashes such that similar images have the same hash. This allows Apple to detect if abusive images are on a device. The hashes are intended, however, to prevent Apple from learning anything not related to abusive images.

We illustrate how systems that attempt to provide privacy using LSH fail to achieve their privacy objectives. In particular, LSH hashes leak information about its input, since the hashes do not provide security properties like pre-image resistance. None of the referenced works, however, were concerned by the privacy implications of this information leakage or considered its seriousness. We therefore investigate the severity of the leakage by developing attacks on two recent applications: FLoC [27] and the MinHash Hierarchy system [9].

FLoC is a system for private targeted advertising, proposed by Google. It uses SimHash to cluster users so that users with similar browsing histories are likely to be in similar cohorts. It aims at providing k-anonymity [33] while computing cohorts useful for targeted advertisements.

MinHash Hierarchy is a system for analyzing traffic trajectories. It computes statistics on location trajectories of mobile devices for urban planning, while ensuring differential privacy for these devices. It works by having cell stations store hashes that represent subsets of all mobile devices passing by.

In this paper, we present three kinds of critical attacks on FLoC, which we illustrate using the MovieLens dataset [14]. First, we present a pre-image attack on SimHash using integer programming. We then design a Sybil attack [10] that generates dozens of histories per second whose hash matches the target hash. We demonstrate how this attack breaks FLoC's k-anonymity, and hence we can identify individuals in cohorts. Furthermore, using Generative Adversarial Networks (GANs) [12, 13], we partially reconstruct plausible histories from just the target hash. With this attack, we show how to break FLoC's privacy claims and infer some of the websites visited by users. Specifically, we can reconstruct 10% or more of the history of at least 30% of the users. Although FLoC is no longer used by Google, these attacks highlight the privacy limitations of LSH-based systems.

Proceedings on Privacy Enhancing Technologies 2023(4), 117–131

https://doi.org/10.56553/popets-2023-0101

In the context of the MinHash Hierarchy, we demonstrate using taxi trajectories in the city of Porto,[1] that we can decide for some individuals whether they followed a particular trajectory, violating their differential privacy guarantee. Our attack narrows down the taxi drivers' trajectories, on average, to around 10% of the city's area, which corresponds to a neighborhood of Porto.

In both of these proposals, our attacks show that if the hashes can measure the similarity of objects, then they also contain fingerprints of the objects themselves. The amount of data in this fingerprint is bounded by the hash size, and while larger hashes provide more utility, they also contain more sensitive information. We are the first to evaluate this information leakage by attacking proposals of significant importance and measuring the information that attackers gain. Implementations of our attacks and more information including an extended report about FLoC are available at https://karelkubicek.github.io/post/floc. Although some countermeasures have been proposed, we discuss in Section 8 how they fail to prevent our attacks.

We emphasize that our main contribution extends beyond our attacks. Namely, we argue that privacy systems based on LSH are generally vulnerable to attacks that allow the extraction of sensitive information from the hashes, which are available to adversaries within these systems by design. Furthermore, even stronger approaches to privacy like differential privacy are still insufficient in addressing these limitations without significant system redesign, as we explain in Section 8. Our work aims at increasing awareness about the vulnerabilities of LSH-based privacy systems and highlighting the importance of developing tools that can rigorously certify the privacy properties of these systems.

Overall, **our contributions** are the following. First, we present a **pre-image attack** on SimHash using integer programming. Second, we implement practical **Sybil attacks** on FLoC by finding pre-images of a target SimHash. We show that this breaks the k-anonymity promised for FLoC by isolating specific users in a cohort. Third, using GANs, we implement a **privacy attack** that reconstructs more than 10% of the browsing history of at least 30% of users based only on the FLoC hash. We also show how to amplify this attack to increase the size of the reconstructed history by exploiting changes in users' SimHash. Finally, for the MinHash Hierarchy system, we present **privacy and pre-image attacks** that identify a subset of the individuals that visited a given checkpoint. We also show that we can track users to narrow down their trajectory to an average of 10% of the city area, which corresponds to a local neighborhood.

2 FEDERATED LEARNING OF COHORTS

In this section, we give some preliminaries on locality-sensitive hashing (Section 2.1). Afterwards, we present SimHash, a class of LSH, and FLoC, a proposal for privacy-preserving targeted advertisements (Section 2.2).

2.1 SimHash

Locality-sensitive hashing (*LSH*) is a class of hash functions mapping similar inputs to similar outputs [26]. These hash functions are usually neither collision nor pre-image resistant like cryptographic hash functions.

In this section, we explain locality-sensitive hashing (*LSH*) and SimHash, a popular instance proposed by Charikar [7] and used by FLoC for privacy-preserving targeted advertising. In particular, Google researchers used SimHash to detect near duplicate websites with the search crawler Googlebot and more recently to measure the similarity between two browsing histories in FLoC. We explain next how SimHash works in the context of browsing histories.

We describe how to compute the SimHash of length ℓ of a browsing history D, which we represent as a finite set of domains. First, we produce for each domain $d \in D$ a *fingerprint vector*, which is a vector $\eta_d \in \mathbb{R}^\ell$ sampled from the standard multivariate Gaussian in $\mathbb{R}^\ell$ using a pseudo-random generator that takes d as the seed. Then we compute $y^{(D)} = \sum_{d \in D} \eta_d$ and the SimHash is $z^{(D)} = \text{sgn}\left(y^{(D)}\right)$, where sgn applies the sign function elementwise to each entry of $y^{(D)}$. Note that the SimHash $z^{(D)} \in \{0,1\}^n$ is a binary vector.

We now give an intuition of why the SimHash is locally sensitive. Suppose that D and D' are two browsing histories of the same size that differ in only one element. Then the sets of fingerprint vectors for D and D' differ in at most one vector. As a result, the sum $y^{(D)}$ of fingerprint vectors in D is probably similar to the sum $y^{(D')}$. Therefore, $z^{(D)}$ and $z^{(D')}$ are probably the same. Note that the greater the number of different elements that D and D' have, the less likely it is that $z^{(D)} = z^{(D')}$.

We illustrate this computation for $\ell = 5$ in Table 1. We have two browsing histories with three domains and a 5-bit target SimHash. The two browsing histories only differ in one domain and the resulting SimHash values only differ by one bit. Note how a slight change in the input changed only one bit of the resulting SimHash.

Table 1: Example of a SimHash computation.

Hist. 1	Domain fingerprints η_d					Hist. 2	Domain fingerprints η_d				
google	2.03	0.18	0.67	0.62	-0.88	google	2.03	0.18	0.67	0.62	-0.88
youtube	-1.51	-1.79	-0.26	0.76	1.11	youtube	-1.51	-1.79	-0.26	0.76	1.11
facebook	0.07	-0.03	-1.55	-0.62	1.61	netflix	0.46	0.67	0.20	-1.24	0.03
sum:	0.59	-1.64	-1.14	0.76	1.84	**sum:**	0.98	-0.94	0.61	0.14	0.26
sign:	1	0	**0**	1	1	**sign:**	1	0	**1**	1	1

2.2 Application of SimHash to Privacy

In this section, we present *Federated Learning of Cohorts* (FLoC) [27]. FLoC is a proposal from Google researchers to partially replace third-party cookies and perform privacy-preserving targeted advertisements. The idea is that users are grouped into cohorts so that users with similar browsing histories are assigned to the same cohort. Each cohort is then assigned an identifier. Instead of revealing personal browsing histories to advertisers, only the cohorts' identifiers are revealed.

2.2.1 Clustering of SimHashes. The FLoC proposal states that a SimHash is computed in the client's browser and serves as a history fingerprint, and only the hash is shared with a central clustering server. This server assigns each user a cohort identifier, where a cohort is a cluster of users with similar SimHashes. We illustrate the clustering procedure in Fig. 1 and describe it below.

[1] Porto is the second largest city of Portugal with 232 000 citizens occupying 41 km^2. Our dataset covers roughly 80 km^2, since it includes the surrounding urban area.

SimHash	Cohort ID
00011	1
01101	1
10000	2
10011	2
10110	3
10111	3
11001	4
11110	4

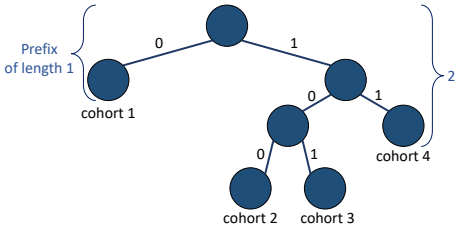

Figure 1: Example of a clustering with FLoC

Let $\mathcal{D}$ be the set of browsing histories of a given set of users. For a bitstring $\sigma \in \{0,1\}^*$ and $\prec$ the prefix operator on strings, let $C_\sigma = \left\{D \in \mathcal{D} : \sigma \prec z^{(D)}\right\}$; that is, C_σ contains all users (i.e., browsing histories) whose SimHash has σ as a prefix. We call C_σ a *cohort* and we say that a cohort is *k-decomposable*, for $k \in \mathbb{N}$, if $|C_{\sigma 0}| \geq k$ and $|C_{\sigma 1}| \geq k$. The clustering procedure starts with a clustering $C = \{C_\epsilon\}$, where ϵ is the empty bitstring; that is, there is only one cluster at the start containing all users. Then a value $k \in \mathbb{N}$ is fixed. As long as there is a k-decomposable cluster $C_\sigma \in C$, the procedure replaces C_σ with $C_{\sigma 0}$ and $C_{\sigma 1}$. The idea is that each cohort in C provides k-anonymity, while containing a set of users with similar browsing histories.

Fig. 1 illustrates the result of the clustering procedure on a set of eight users. The table given there shows the SimHash of the browsing history of each user and an identifier of the cohort to which they have been assigned by the clustering procedure. Note how each cohort has $k = 2$ users. The tree in the figure illustrates how the clustering procedure divided 2-decomposable cohorts until reaching the clustering assignment depicted in the table.

2.2.2 Origin Trial. From March 30 to July 13 2021, Google tested FLoC in its Origin trial [25]. Users of Chrome version numbers 89 - 91 located in ten countries were eligible for the experiment. Only 0.5% of these eligible users took part in the Origin trial, and only websites that requested a FLoC ID were added to the history used for FLoC computation. 50-bit SimHashes were computed on a domain history of one week. Out of the 50 bits, only 13 to 20 bits were necessary to split the users into around 33 000 cohorts of at least 2000 users.

Despite the small user sample, some advertisers were successful in identifying topics of interest for users in the cohorts. For example, we refer to Criteo's blog [29] for the evolution over time of a cohort with around 10 000 users. We summarize their world cloud representation of the most popular topics in Table 2, showing just the five main topics. Also note that the main topics would vary much more in the case where only a small number of users can be observed.

CafeMedia, an ad management service, also analyzed the quality of the cohorts for targeted advertising [23]. They formed groups of 1000 cohorts and computed the most frequent 10 keywords occurring in the browsing histories in those groups. Table 3 shows the top 10 keywords for 5 groups. They could, for example, distinguish groups that are more interested in business and professional development and also groups that are more interested in leisure activities.

Table 2: Criteo's example of the evolution of the most frequent topics browsed by a large cohort (≈ 10 000 users)

Week 0	Week 3	Week 5
Gaming	Gaming	Tech. & Computing
Tech. & Computing	Tech. & Computing	Gaming
Books and Literature	Education	News and Politics
Education	Shopping	Style & Fashion
Shopping	News and Politics	Healthy Living

Table 3: CafeMedia's extracted interest keywords of the selected cohorts (see the complete table in [23])

Cohort IDs	Keywords				
0-1k	music	support	grade	questions	season
1k-2k	dogs	guides	working	things	roast
2k-3k	writing	magic	vegetables	movies	slow
3k-4k	prime	high	rolls	magic	chili
4k-5k	weekly	world	disney	magic	sheets

Table 4: Attack summary table

Attack Name	Privacy Properties	Type of Attack
Pre-image	Pre-image Resistance	Pre-image Attack
Sybil	k-anonymity	Forgery Attack
GAN-IP	User Browsing Privacy	Privacy Attack

3 ATTACKS ON FLOC

In this section, we present attacks that break FLoC's privacy properties. We first present a **pre-image attack on SimHash** (Section 3.1) that breaks its *pre-image resistance property*. In our experiments, the pre-image attack can be used to mount a **Sybil attack** to break its *k-anonymity property* as well (Section 3.2). Using Generative Adversarial Networks (*GANs*) [12, 13], we propose the **GAN-IP attack**, which *recovers parts of the browsing history* of real users, since GANs can be used to generate plausible browsing histories for users in a target cohort (Section 3.3). The GAN-IP attack can reconstruct 10% or more of the history in at least 30% of the cases, breaking FLoC's guarantees of keeping browsing histories private. Table 4 and Fig. 2 summarize and illustrate our three attacks.

We remark that the pre-image attack differs from the GAN-IP attack in that the former only produces histories that match a specific SimHash. Our GAN-IP attack is more powerful because it produces images that resemble real-user histories from a particular target population. The results can be exploited to infer information about the real-user histories, as we show in Section 4.

We give an overview of the GAN-IP attack. Using a GAN we generate plausible user histories, which we give to an integer program. For each history, the integer program finds a non-empty subset of the history that matches the given target SimHash. Fig. 3 illustrates the GAN-IP attack. We can optionally apply the GAN's discriminator on the integer program's output, as shown in the

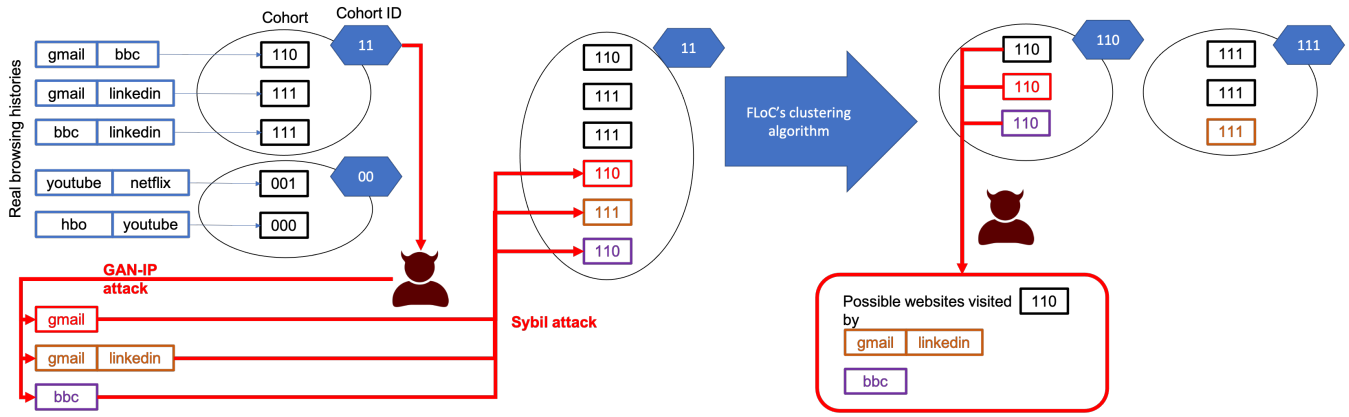

Figure 2: How an attacker extracts private information from FLoC. First, the attacker takes a cohort ID γ and then uses the GAN-IP attack to create fake browsing histories whose SimHash contains γ as prefix. These SimHashes make the cohort decomposable, so FLoC's clustering algorithm divides the cohort into smaller cohorts. The attacker exploits the fake histories to infer websites visited by real users.

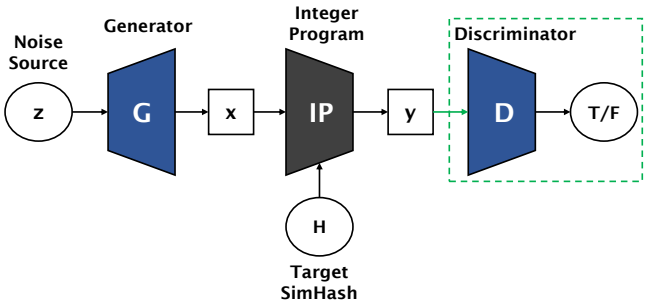

Figure 3: Pipeline: integer programming on the generator outputs. The green boxed part is optional.

green frame in Fig. 3. In this way, the discriminator gives us a score on how realistic the produced browsing history is.

Attacker model. We assume that the attacker's goal is to infer private information about the browsing history of a target user. For that, we assume the following capabilities. (1) The attacker has access to the FLoC implementation used by the users' devices. This is trivial since the code is embedded in the open-source Chrome browser. (2) The attacker can see the target user's FLoC ID, which the user sends to all websites embedding a FLoC request. (3) The attacker can actively create new users in the FLoC system. This is possible because the server that assigns cohort IDs takes as input only the SimHash and, therefore, it cannot distinguish genuine users from bots. Note that common bot-detection techniques such as CAPTCHAs are not suitable, as they would sacrifice usability by requiring users to solve CAPTCHAs to receive targeted advertisements. Also, privacy-invasive techniques like requiring a Google account or fingerprinting would defeat FLoC's purpose as a privacy mechanism. (4) The attacker has access to the browsing histories of a sample of the user population, which can also be purchased from companies such as Comscore. Examples of such attackers, ordered by increasing capabilities, are operators of any website, tracking websites, and also Google itself.

We used the SimHash implementation from Chrome Version 91 according to capability 1. Our Sybil attack depends on knowledge of the target user's cohort ID (capability 2) and the ability to generate new users (capability 3). We train the model used for the GAN-IP attack on a publicly available dataset of movies, which the FLoC authors also used for evaluation. There also exists proprietary datasets of browsing histories that can be used in the real attack (capability 4).

3.1 Integer Programming Pre-image Attack

We now show how to compute pre-images of SimHashes using integer linear programming. Assume given a set $D = \{d_1, \ldots, d_n\}$ of domains (e.g., output by the GAN's generator) and a SimHash $z \in \mathbb{R}^\ell$ and we want to find a subset $D^* \subseteq D$ whose SimHash is z. We start by observing that D^* must fulfill the following condition, by the definition of SimHash,

$$\operatorname{sgn}\left(\sum_{d \in D^*} \eta_{d,j}\right) = z_j, \quad \text{for } j \leq \ell, \tag{1}$$

where $\eta_{d,j}$ is the j-th entry of η_d. If we unfold the definition of sgn, this condition becomes: for $j \leq \ell$, $\sum_{d \in D^*} \eta_{d,j} \geq 0$, if $z_j = 1$, and $\sum_{d \in D^*} \eta_{d,j} < 0$, otherwise. We can rewrite this condition to:

$$(2z_j - 1) \sum_{d \in D^*} \eta_{d,j} \geq 0, \text{for } j \leq \ell. \tag{2}$$

We see that finding a pre-image of the SimHash z reduces to finding a subset $D^* \subseteq D$ that fulfills these ℓ inequalities. We next show how to do this using integer programming. We first represent subsets of D as bitstrings in $\{0,1\}^n$. A bitstring $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ denotes the subset $\{d_i : i \leq n, x_i = 1\}$. If x^* is the bitstring representation of D^*, we can then rewrite the condition as:

$$(2z_j - 1) \sum_{i \leq n} \eta_{d,j} x_i \geq 0, \text{for } j \leq \ell. \tag{3}$$

Table 5: Benchmark of GAN – Integer Program

SimHash Length	Success Rate	Int. Program Time
5	100%	0.52 s
10	95%	2.01 s
15	64%	5.03 s
20	34%	5.89 s
25	11%	12.83 s

This leads to the following linear integer program:

$$\max_{x} \quad \sum_{i \le n} x_i \tag{4}$$

$$s.t. \quad (2z_j - 1) \sum_{i \le n} \eta_{d,j} x_i \ge 0, \text{ for } j \le \ell \tag{5}$$

$$x_i \in \{0, 1\}. \tag{6}$$

Note that by maximizing $\sum_{i \le n} x_i$, we seek the largest subset $D^* \subseteq D$ that fulfills the conditions. Hence, this program searches for the largest subset of D that yields the desired SimHash z. The maximization is also necessary to avoid outputting $x = 0^n$, which is a trivial solution. We summarize these insights with the following theorem.

Theorem 1. Assume given a SimHash z and the integer program $IP(D)$ above. If $x^* \in \{0,1\}^n$ is an optimal solution to the program below and $x^* \neq 0$, then the SimHash of x^* is z.

As illustration, we present the integer program with a history D containing exactly `google.com`, `youtube.com`, and `facebook.com`. This is the history in the left of Table 1 As a target SimHash, we choose the SimHash of the right history (**10111**). We get the following integer program. We maximize $\sum_{i \le 3} x_i$ with the constraints

$$\begin{aligned}
(2 \cdot \underline{\mathbf{1}} - 1) \cdot (-0.88 \cdot x_1 + 1.11 \cdot x_2 + 1.61 \cdot x_3) &\ge 0 \\
(2 \cdot \underline{\mathbf{1}} - 1) \cdot (0.62 \cdot x_1 + 0.76 \cdot x_2 - 0.62 \cdot x_3) &\ge 0 \\
(2 \cdot \underline{\mathbf{1}} - 1) \cdot (0.67 \cdot x_1 - 0.26 \cdot x_2 - 1.55 \cdot x_3) &\ge 0 \\
(2 \cdot \underline{\mathbf{0}} - 1) \cdot (0.18 \cdot x_1 - 1.79 \cdot x_2 - 0.03 \cdot x_3) &\ge 0 \\
(2 \cdot \underline{\mathbf{1}} - 1) \cdot (2.03 \cdot x_1 - 1.51 \cdot x_2 - 0.07 \cdot x_3) &\ge 0.
\end{aligned}$$

The optimal solution is $(x_1, x_2, x_3) = (1, 1, 0)$. We conclude that the `facebook.com` domain of the history in the left-hand side of Table 1 must be removed to match the target SimHash for the history on the right-hand side. This means that the `netflix.com` domain of the right history is redundant, since it does not change the SimHash of the remaining domains.

Although finding a pre-image of SimHash is NP hard, our integer programming attack is very efficient for the used bit lengths and history sizes, as illustrated in Table 5. We vary the SimHash bit length from 5 to 25 in increments of 5. For 8 billion people, a SimHash length of 25 bits yields average cohort sizes of less than 240 users. Such cohorts would be invalid as they do not respect the minimum size of at least 2000 users required by FLoC. Therefore, SimHash prefixes longer than 22 bits are unlikely to be used. This is why the SimHash length in FLoC trials varied from 13 to 20 bits. For a given SimHash length, we sample a real history and compute its corresponding SimHash. The integer program then starts with a history D of 32 elements, which can be either random or generated using a GAN introduced in Section 3.3.

In Table 5, the second column reports the percentage of histories generated by the GAN for which we could find a subset matching the target SimHash. We also report the average runtime in the third column. These results are based on executions on four different histories of real users generating at least 25 pre-image histories with the same SimHash.

This demonstrates that it is very efficient to find pre-images for a target SimHash. This facilitates creating fake users and inferring private information about the browsing history of real users.

3.2 Sybil Attack

The privacy goals of FLoC is to achieve k-anonymity for the users [27]. A Sybil attack floods a system with real users by generating fake (Sybil) entities. We show how integer programming can be used to mount a Sybil attack to deanonymize users hiding in clusters. The Sybil attack can isolate and identify users in a cohort, breaking the k-anonymity property of FLoC.

Let u be a target user, let z be the SimHash of their browsing history, and let σ be u's cohort ID, that is, the prefix σ of z such that z is contained in the cohort C_σ computed by FLoC's clustering procedure (see Section 2.2.1). Note that σ is known to FLoC. We show next how to infer additional information about z.

We first generate some arbitrary bitstrings $z_1, \ldots, z_M$ such that $\sigma \prec z_i$, for all $i \le M$. For each z_i, we use our pre-image attack to generate a set S_i of fake user histories whose SimHash is z_i; we call these the *Sybil users*. Let $S_\sigma = \bigcup_i S_i$ be the union of these histories.

When these histories are sent to the clustering procedure, the procedure adds them to C_σ, as σ is a prefix common to all strings in S_σ. By making S_σ sufficiently large, we make C_σ decomposable (see Section 2.2.1). The clustering procedure will then divide C_σ into $C_{\sigma 0}$ and $C_{\sigma 1}$. If we then look at u's cohort ID, we see that it changes from σ to either $\sigma 0$ or $\sigma 1$, revealing one extra bit of u's SimHash. Furthermore, since we own the Sybil users and know their SimHashes, further iterations of this process will reveal increasingly many bits of u. After sufficiently many repetitions (approximately $\log_2(k) \approx 11$ iterations), σ becomes a longer string σ' so that $C_{\sigma'}$ contains only u and a subset of our Sybil users. This breaks the k-anonymity of FLoC.

In Fig. 4, we demonstrate this attack on a toy example. At timestamp 1, we have the cohorts C_0 and C_1. The minimum size for a cohort is $k = 2$. Mounting a Sybil attack to extend the prefix length of some cohorts, we generate two fake Sybil users, which are assigned to cohort C_0. The new Sybil users make C_0 k-decomposable, so the clustering procedure partitions C_0 into C_{01} and C_{01}. Observe that C_{01} consists now of two Sybil users and one real user. So the attacker can approximate the browsing history of that user using the generated browsing histories of the Sybil users.

3.3 GAN-IP Privacy Attack

The browsing histories generated by our integer program may not resemble a browsing history produced by a human. To produce a more realistic distribution of browsing histories and to gain more insights on the histories hidden in a cohort, we combine GANs with our integer programming attack to produce the GAN-IP attack.

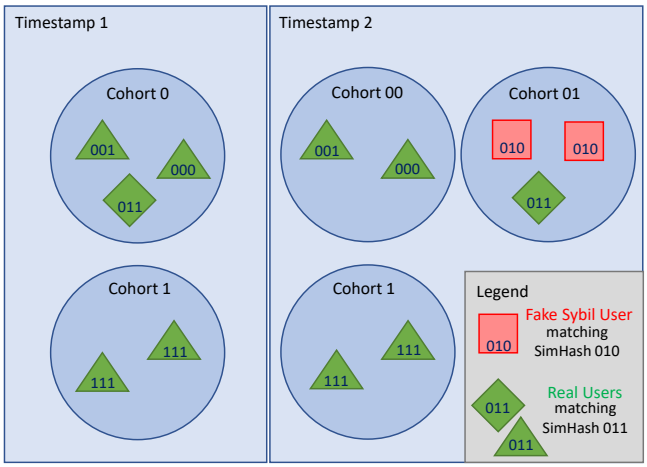

Figure 4: Sybil attack example

Generative Adversarial Networks (GANs) can generate new samples from the same distribution as the training data. A GAN consists of two neural networks, a generator G and a discriminator D. They compete against each other during training. The generator learns to produce realistic samples with the objective of deceiving the discriminator, while the discriminator learns to differentiate between the generated and real samples. From the implementations available, we chose LeakGAN [13] because it is designed for text generation. However our attack should work with any GAN that can be adapted to produce users' histories.

We now present the GAN-IP attack. Suppose that we are given a SimHash z of a given browsing history h and that we want to produce a set H of histories whose SimHashes are all equal to z. First, we use the LeakGAN to produce a set H' of histories that resemble a sample from the distribution of browsing histories. Then for each $f \in H$, we attempt to compute a solution x_f^* of $IP(f)$, the integer program induced by f. The desired set H is $\left\{x_f^* : f \in H, IP(f) \text{ has a non-trivial solution}\right\}$. The attack is illustrated in Fig. 3.

To summarize, we can combine our three attacks to extract private information as follows. First, we use a GAN to learn a distribution of users' histories such that, in approximately 30% of the cases, the generated user will share 10% or more of the history with the target user in the cohort. Using the GAN's generator, we then produce fake browsing histories that resemble histories from real users. Afterwards, we compute from this generated history a subset that matches a particular SimHash prefix of a target cohort using the IP-attack. These matching histories allow us to mount a Sybil attack, breaking not only the k-anonymity of users, but extending the prefix length used to assign the cohort. This leaks more of the users' SimHash, forming a self-reinforcing loop for the IP-attack, inferring parts of the users' browsing history.

4 ATTACK IMPLEMENTATION FOR FLOC

For data protection reasons, we do not have access to a public browsing history dataset. To evaluate our attacks, we instead use the MovieLens dataset [14]. An entry in this dataset contains movies watched by users over a period of time. Note that a movie history reflects a user's preferences and can be used to generate movie recommendations for that user. For these reasons, the MovieLens dataset acts as a good proxy to evaluate how our attacks would work in real browsing histories. We also remark that FLoC's whitepaper also used this same dataset to evaluate FLoC [27].

We launch our GAN-IP attack on different movie histories from the MovieLens dataset. We demonstrate that the movie histories produced by our GAN-IP attack contain on average at least 10% of the movie histories targeted by our attack. Furthermore, in about 50% of our tests, the movie histories produced by the GAN component alone contain at least 10% of the targeted histories. This demonstrates that the GAN-IP attack can extract information that was intended to remain private by the FLoC system.

4.1 Setup

For demonstration purposes, the GAN was trained to generate movie histories consisting of at most 32 histories and using only the 5000 most watched movies. However, our attack can be extended to larger movie histories and larger sets of movies. The GAN model can only predict domains that are present in the training data. If the training data is too large, contextualized models can be created, taking into account additional information sources to more precisely target specific populations. In the complete GAN-IP attack, new domains of interest to the attacker can be added to the integer programming part. We divided the MovieLens dataset into a training set and a test set. The training set contains 120 000 histories and the test set 5000 histories. The LeakGAN used by the GAN-IP attack was trained for 12 hours using an Nvidia GeForce RTX 2070 Super GPU.

We chose the values for the maximum history length (32), token size (5000), and train-test split for the GAN to be similar to the settings of the LeakGAN's experiments. Higher values significantly increase training and generation times. For comparison and reproducibility purposes, we employed the LeakGAN model as is, adhering to software package requirements from 2017. Note that over the past five years there has been considerable progress in software and hardware for training machine learning models. Therefore, with updated software libraries and hardware, we expect the existing limitations to be significantly mitigated.

We evaluate our GAN-IP attack with 5 movie histories sampled from the test set. For each movie history h_i, with $i \leq 5$, we compute its SimHash s_i and give it as input to the GAN-IP attack, which produces a set of movie histories $\hat{H}_i$ whose SimHash is also s_i. The set $\hat{H}_i$ contains at least 200 histories and s_i is 15 bits long. We evaluate the quality of $\hat{H}_i$ with $I_i = \frac{1}{|\hat{H}_i|} \sum_{\hat{h} \in \hat{H}_i} \left|\hat{h} \cap h_i\right|$, the average number of movies that the generated histories of the GAN-IP attack have in common with the target history h_i. The quality of our attack is then $q := \frac{1}{5} \sum_i I_i$. When reporting q, we also report the standard deviation of $I_1, \ldots, I_5$. We measure q on various GAN models. As a baseline, we can use a random generator instead of the GAN's generator. Note that q indicates how much of the browsing history generated by our attack can be used to infer the movie history of a user with the same SimHash. Hence, our attacks shall maximize this value q.

Table 6: Distribution of Common Movie Counts

Generator	Common Movies ± stdev (% of Gen. History Len.)	
	Generator	Int. Prog.
RAND	0.20 ± 0.04 (< 1%)	0.17 ± 0.03 (≈ 1%)
GAN-41	**2.39 ± 0.96** (≈ 9%)	**1.77 ± 0.90** (≈ 12%)
GAN-61	1.93 ± 0.71 (≈ 7%)	1.33 ± 0.58 (≈ 9%)

4.2 Results

The GAN-IP attack can extract sensitive information from the SimHash. Table 6 reports q for three versions of the GAN-IP attack: RAND, which uses only a random generator instead of a GAN to produce the set H' of histories; GAN-41, which uses LeakGAN's weight from the saved training iteration 41; GAN-61, which is analogous to GAN-41 except for being trained for 61 iterations. In parenthesis we give on average (in percentage) the part of the full generated history in common with the target history. For the GAN generators, the average history length is approximately 27 and 15 after the integer program. For the random generator the values are 32 and 17. This sets the upper bound on the number of common movies, since the histories filtered by the integer program are only about half of the maximal length. Observe how GAN-41 produces higher values of q than GAN-61 and RAND. Hence, stopping the training at iteration 41 yields histories with more movies in common with the target history.

The use of GANs significantly improves the attack's quality. To demonstrate that the GAN-IP attack provides significant information, we compare the movies in the histories produced by RAND, GAN-41, and GAN-61. Fig. 5 is a histogram that shows, for $n \leq 11$ and each version of the GAN-IP attack, how many histories h were produced such that $|h \cap h_i| = n$, for some $i \leq 5$. The number of movies that our generated histories have with the target histories is between 0 and 11. Observe how GAN-41 and GAN-61 in comparison with RAND have higher common movie counts with the target history. Therefore, histories generated with these GANs leak on average more information about the target history.

In Fig. 6, we present an analogous histogram, but for the history produced only by the GAN. That is, we take the history h' produced by the GAN before it was passed to the integer programming to produce the history $\tilde{h}$.

In Table 8, we show an example of a movie target history, a history generated by the GAN, and a history generated by our GAN-IP attack. The real history h (on the left) is from the test set. We computed its SimHash and then generated a set of fake movie histories using the GAN. The history h' in the middle is an example of such history. We gave this history as input to the IP-attack and then generated the history $\tilde{h}$ on the right. The SimHash of $\tilde{h}$ matches the SimHash of h and 50% of its movies are from h. In blue we show the movies that h and h' have in common.

4.3 Discussion

From the histograms we see that the random generation has few movies in common with the target history. However, our GAN model evaluated at two different checkpoints has many more histories with a higher number of common movies. This is promising

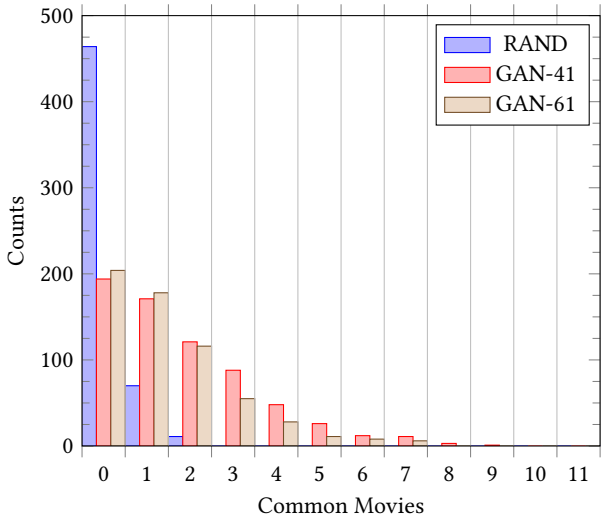

Figure 5: Histogram of common movie counts between h and $\tilde{h}$ (with integer programming)

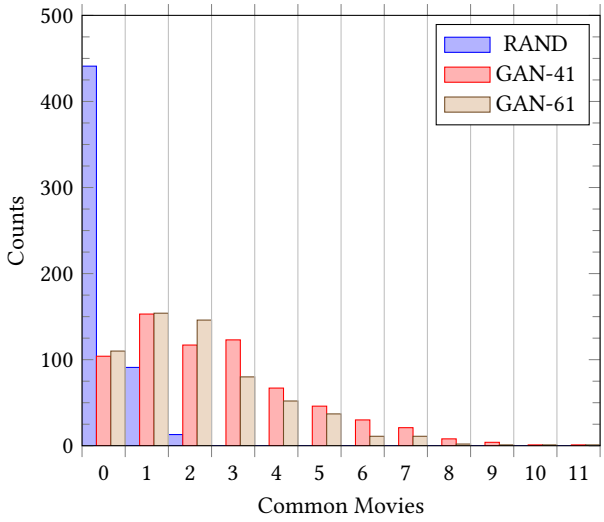

Figure 6: Histogram of common movie counts between h and h' (without integer programming)

but only the tail of the distribution is on the higher counts, with a maximum of 9 common movies for one history of GAN-41 in Fig. 5. In Fig. 6 both GAN-41 and GAN-61 have one history with 11 and 10 common movies with the target. Moreover, GAN-41 has 4 histories with 9 common movies while GAN-61 still only has one. On average, the number of common movies with a target history is around 2 (see Table 6).

The histories generated by GAN-41 and filtered by integer programming have on average 12% movies in common with the target histories. In around 28% of the cases, the subset of movies selected by the IP attack matching the target SimHash reconstructs more

than 10% of the target history. Hence, the GAN-IP attack successfully breaks FLoC privacy claims and infers parts of the target user's history.

Although the overall accuracy of our attacks is not very high, our attacks are limited to processing a single iteration of FLoC at one time stamp. However, if an attacker were to execute our attack over an extended time period, the results could be significantly amplified. While the SimHash of users likely changes with each iteration (once per week), the majority of the browsing patterns, like a user's favorite websites, persists. Consequently, if our attack consistently generates the same website across multiple weeks, it becomes more probable that the user has indeed visited that website. Similarly, the union of sets of generated websites is more likely to encompass all the websites that the user likes to visit, compared to a single browsing history alone. Note that we did not specifically assess the extent of amplification, as the MovieLens dataset varies from actual website histories in that it does not reflect the tendency of users to frequently revisit their popular websites.

Since Google runs the clustering algorithm, it is ideally suited to perform the GAN-IP attack. We therefore note some of Google's capabilities that can make the attack more efficient. First, Google collects anonymized browsing histories of Chrome users that agreed with data collection in the Chrome User Experience Report. This gives them a significantly larger dataset of real browsing histories compared to the MovieLens dataset that we used. Second, Google has significantly more computational resources. Therefore, our results should be viewed only as a lower-bound of what a more powerful adversary can achieve.

The generated history shares on average a non-negligible percentage of common movies with the unseen target history. The attack thus succeeds in revealing potentially sensitive information about the target user and, by extension, sensitive information about other users in the same cohort. Our attacks therefore break the k-anonymity and history-privacy properties claimed by FLoC.

5 MINHASH HIERARCHY

In this section, we give some preliminaries on MinHash, a class of LSH (Section 5.1). Afterwards, we present the MinHash Hierarchy system, a proposal for computing statistics on vehicles' trajectories (Section 5.2). We then present our pre-image attack on the MinHash Hierarchy system (Section 6).

5.1 MinHash

MinHash is a type of LSH proposed by Broder [6]. For a set of objects $\mathcal{X}$, MinHash estimates the similarity of subsets of $\mathcal{X}$. A MinHash is a function h that maps each subset X of $\mathcal{X}$ to a pseudo-random sequence $h(X) = (s_1, \ldots, s_n)$ of n bitstrings. Usually, these bitstrings are 32 bits long. The function h has the following property: for any $Y \subseteq \mathcal{X}$, the probability of $h(X) = h(Y)$ is the Jaccard similarity between X and Y, i.e., $\frac{|X \cap Y|}{|X \cup Y|}$.

A MinHash function h is composed of n hash functions $h_i : \mathcal{X} \to \mathbb{N}$ and the MinHash of $X \subseteq \mathcal{X}$ is $h(X) = (s_1(X), \ldots, s_n(X))$, where $s_i(X) = \min_{x \in X} h_i(x)$.

A common choice for each $h_1, \ldots, h_n$ builds upon a hash function $\pi : \mathcal{X} \to \{0, \ldots, 2^{32} - 1\}$ that maps $\mathcal{X}$ to the set of 32-bit strings. Then, for $i \leq n$, $h_i(x) = r \cdot \pi(x) + c \mod p$, where r, c, and p are chosen uniformly at random from a sufficiently large interval of natural numbers and p is a prime number greater than $\max\{\pi(x) : x \in \mathcal{X}\}$ [6].

We illustrate the computation of a MinHash signature in a simple example. We define three hash functions $h_1(x) = x + 3 \mod 5$, $h_2(x) = 2x + 1 \mod 5$, and $h_3(x) = 3x + 4 \mod 5$. Let $\mathcal{X} = \{0, 1, 2, 3, 4\}$. We now compute the MinHash signature $h(X)$ for the set $X = \{1, 4\}$. Note that $s_1(X) = \min\{h_1(1), h_1(4)\} = 2$, $s_2(X) = \min\{h_2(1), h_2(4)\} = 3$, and $s_3(X) = \min\{h_3(1), h_3(4)\} = 1$. Hence, the MinHash signature for the set X is then $(2, 3, 1)$.

5.2 Application of MinHash to Privacy

In this section, we present the MinHash Hierarchy [9], which is a proposal for computing statistics on mobile entities' location trajectories. One example of such a statistic is the most popular route in the city. The MinHash Hierarchy can compute such statistics by placing cellular base stations, called *checkpoints*, in a city and assigning a bitstring to each vehicle. Each checkpoint collects the set X of bitstrings of the vehicles that pass nearby, using mobile devices stored in the vehicle. Afterwards, each checkpoint stores *a MinHash signature*, which is the MinHash of X.

We mainly focus on the MinHash aspect of the MinHash Hierarchy and we therefore simplify its exposition.

5.2.1 MinHash Signatures. We present here how the MinHash signatures are computed. Let n be the number of vehicles driving in a city. First, $m \ll n$ checkpoints are distributed throughout the city. Then $k \in \mathbb{N}$ hash functions $h_1, \ldots, h_k$ are fixed. The recommendation is to let $h_i(x) = ax + b \mod p$, with $i \leq k$, $a, b, p \in \mathbb{N}$, and $p > n$ prime, as shown before. However, if needed, cryptographic one-way functions can be used instead.

Each checkpoint maintains a MinHash signature $s = (s_1, \ldots, s_k)$ so that, at any time, s is the MinHash of the set of vehicles that passed by the checkpoint so far. To ensure this, s_i is initially set to ∞, for $i \leq k$, as the MinHash of the empty set is $(\infty, \ldots, \infty)$. Next, whenever a vehicle whose assigned bitstring is x passes by the checkpoint, s_i is updated to $\min(s_i, h_i(x))$, for $i \leq n$.

With the checkpoints' MinHash signatures, Ding et al. [9] proposed the MinHash Hierarchy to efficiently perform common path queries, such as finding the most frequented roads in a city during a given time interval. The process uses intersection and union operations defined for MinHash signatures of checkpoints to estimate the Jaccard similarities. Our attack focuses on the MinHash signatures and should work irrespective of the operations used to derive a given MinHash signature.

5.2.2 Wrong Differential Privacy Claim. Ding et al. [9] claim in Theorem 5.1 that the MinHash Hierarchy provides differential privacy for the vehicles. We show that this claim is wrong. We start by recalling the definition of differential privacy. An algorithm A is ϵ-differentially private if for any of A's possible outputs O and for all databases D_1 and D_2 that differ in only one individual [11]:

$$P[A(D_1) = O] \leq e^{\epsilon} \cdot P[A(D_2) = O]. \tag{7}$$

In our context, a database D is a set of location *trajectories*, each individual is a trajectory, and the Algorithm A is the one used by a checkpoint to compute its MinHash signature. For simplicity and

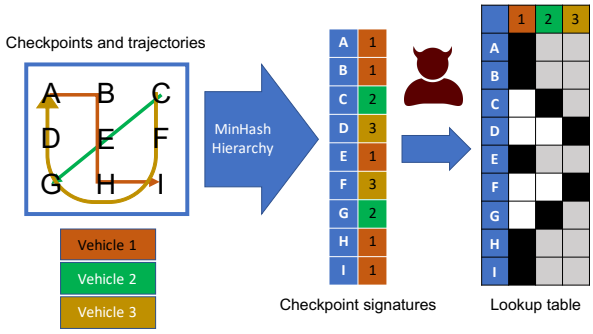

Figure 7: How an attacker extracts private information from the MinHash Hierarchy. Left: Checkpoints A–I located in a grid and three vehicles' trajectories. Right: The checkpoints' signatures (we assume only one hash function: the identity function). The attacker computes a partial lookup table that states for each vehicle and each checkpoint, whether the vehicle passed by that trajectory (black: passed, white: not passed, gray: unknown). The lookup table reveals some checkpoints that were visited by some vehicles.

without loss of generality, we can assume a MinHash length of $k = 1$; so there is only one single hash function h.

We refute Theorem 5.1 from [9] with the following counterexample. Let $D_1 = \{t_1, \ldots, t_n\}$ and let $D_2 = D_1 \setminus \{t_n\}$. Suppose that $h(t_1) > \cdots > h(t_n)$. Therefore, $A(D_1) = h(t_n) < A(D_2)$, as $t_n \notin D_2$. Hence, $P[A(D_1) = h(t_n)] = 1$ whereas $P[A(D_2) = h(t_n)] = 0$. Since $e^{\epsilon} > 0$, for any $\epsilon \in \mathbb{R}$, Eq. (7) cannot hold when $O = h(t_n)$.

6 ATTACKS ON MINHASH HIERARCHY

Our counterexample in Section 5.2.2 demonstrates that an attacker with side knowledge can tell if a particular vehicle passed through a particular checkpoint. However, it does not tell us how much information it leaks in practice. Therefore, in this section, we present an attack breaking the privacy properties of the MinHash Hierarchy system that can be used directly to narrow down the area in which a vehicle traveled. In our experiments, we narrowed down the potential trajectory area to 10% of the total area (in the number of checkpoints). The attack is illustrated in Fig. 7.

Attacker model We assume that the attacker wants to infer the trajectories of the vehicles whose data is collected by the MinHash Hierarchy. We also assume that the attacker can access each checkpoint's signature, knows the hash functions used to compute the signatures, and can efficiently compute collisions for them.

Note that the MinHash Hierarchy fulfills the requirements above. In particular, the network operator can access the checkpoint hashes, as they are needed as input to the MinHash Hierarchy. So this system can use our attack to reconstruct trajectories, which is precisely what MinHash Hierarchy tries to avoid. Furthermore, the hash functions used by the MinHash Hierarchy are just permutations. Due to this implementation choice, we can not only compute signature collisions, but we can also invert the permutation, extracting the user identifier. Should it instead use cryptographic hash functions with a large hash length like 256 bits, it would still be possible to precompute a look-up table with the hashes of all users. This is because the input space is the set of all mobile users and the cardinality of this user set is small.

We now present our attacks. Suppose that we are given a vehicle identified with bitstring v and let $z = (z_1, \ldots, z_n)$ with $z_i = h_i(v)$, for $i \leq n$, be *the signature of* v. Let C be the set of checkpoints in a geographical area. For a checkpoint $c \in C$, we denote its signature by $s(c) = (s_1(c), \ldots, s_n(c))$. We use Algorithm 1 to partition C into three subsets B_z (black), G_z (gray), and W_z (white). W_z denotes all checkpoints c such that $z_i < s_i(c)$, for some $i \leq n$. Note that this condition means that the vehicle is not in the set of vehicles that passed through c. Otherwise, $s_i(c) \leq z_i$. B_z contains all the points not in W_z such that $z_i = s_i(c)$, for some $i \leq n$. Note that if $c \in B_z$, then it is very likely, except for a rare hash collision, that the vehicle passed through c. Finally, G_z contains all other checkpoints in C: checkpoints not in W_z for which $z_i < s_i(c)$, for all $i \leq n$. Note that if $c \in G_z$, then it is still likely that the vehicle passed through c, but not as likely as if $c \in B_z$.

Algorithm 1 Attack on MinHash Hierarchy

```
W_z ← ∅
for i = 1, ..., n do
    for c ∈ C do
        if z_i = s_i(c) then
            B_z ← B_z ∪ {c}
        end if
        if z_i < s_i(c) then
            W_z ← W_z ∪ {c}
        end if
    end for
end for
B_z ← B_z \ W_z
G_z ← C \ (B_z ∪ W_z)
return W_z, G_z, B_z
```

Theorem 2. Let v be a vehicle with signature z and let c be a checkpoint.

- If $c \in W_z$ then v cannot have passed through c.
- If $c \in B_z$ then $h_i(v) = h_i(v')$, where $i \leq n$ and v' is some vehicle that passed through c.

Proof. If $c \in W_z$, then $z_i < s_i(c)$, for some $i \leq n$. Recall, by the definition of MinHash, $s_i(c) = \min\{h_i(v') : v' \in V_c\}$, where V_c denotes all vehicles that passed through c. Hence, $v \notin V_c$; otherwise, $s_i(c) \leq h_i(v) = z_i$, which is a contradiction.

For the second claim, note that if $c \in B_z$, then $z_i = s_i(c)$, for some $i \leq n$. By the definition of MinHash, we have $h_i(v) = z_i = s_i(c) = h_i(v')$, for some vehicle $v' \in V_c$. □

We emphasize that the hash functions used by MinHash Hierarchy are not collision-resistant. Even if they use cryptographic hash functions with a large length, the set of mobile users is small enough that one can precompute a look-up table with the hashes of all users. Therefore, if $c \in B_z$, then v is likely to have passed through c.

Observe that $G_z \cup B_z$ describes all possible checkpoints the vehicle could have visited. In our experiments, we found that on average $G_z \cup B_z$ contains only around 10% of all checkpoints in C.

Table 7: Example signatures for vehicles v and checkpoints c

c	$(s_1(c), s_2(c))$	v	$(z_1(v), z_2(v))$
c_1	$(8, 12)$	v_1	$(9, 11)$
c_2	$(6, 3)$	v_2	$(2, 8)$
c_3	$(2, 7)$	v_3	$(12, 13)$
c_4	$(4, 11)$	v_4	$(7, 10)$
c_5	$(11, 5)$	v_5	$(5, 18)$

To illustrate this attack, consider a scenario with 20 vehicle trajectories, 5 checkpoints, and 2 hash functions. The vehicle MinHash is $z = (z_1, z_2) = (9, 11)$. We compare each checkpoint's signature entries to the corresponding vehicle hash. Considering vehicle v_1 and the 5 checkpoints in Table 7, our attack returns $W_z = \{c_1, c_5\}$ because at least one of the hashes is greater in the checkpoint signature. $B_z = \{c_4\}$ because the checkpoint is not in W_z and at least one hash is equal. $G_z = \{c_2, c_3\}$ contains the remaining checkpoints.

Identifying Vehicles

Algorithm 1 takes as input a trajectory and identifies the checkpoints that could have been visited in that trajectory. It is also possible to modify this algorithm so that the input is a checkpoint and the output is the subset of vehicles from a set V that potentially visited that checkpoint. The result is Algorithm 2. This algorithm produces, from a given checkpoint c, three sets of vehicles: W_c, containing the vehicles that certainly did not pass through c; B_c, the vehicles that most likely passed through c (except in the rare case of a hash collision); and G_c, containing the remaining vehicles. For example, suppose that we run this algorithm with checkpoint c_4 as input and with V as the 5 vehicles listed in Table 7. Then $W_c = \{v_2, v_4\}$ since $z_1(v_2) < s_1(c_4)$ and $z_2(v_4) < s_2(c_4)$. $B_c = \{v_1\}$ since $z_2(v_1) = s_2(c_4)$. Finally, $G_c = \{v_3, v_5\}$ contains the remaining checkpoints.

Algorithm 2 Estimating vehicles passing through c

```
W_c ← ∅
for i = 1, ..., n do
    for v ∈ V do
        Compute z = (z_1, ..., z_n) with z_i = h_i(v)
        if z_i = s_i(c) then
            B_c ← B_c ∪ {v}
        end if
        if z_i < s_i(c) then
            W_c ← W_c ∪ {v}
        end if
    end for
end for
B_c ← B_c \ W_c
G_c ← V \ (B_c ∪ W_c)
return W_c, G_c, B_c
```

Theorem 3. Let v be a vehicle.

- If $v \in W_c$ then v cannot have passed through c.
- If $v \in B_c$ then $h_i(v) = h_i(v')$, for some $i \leq n$ and v' some vehicle that passed through c.

The proof is analogous to the previous one. Observe again, that for MinHash Hierarchy, if $v \in B_c$, then v is likely to have passed through c as one can easily precompute a look-up table with the hashes of all vehicles. This theorem shows that we can narrow the set of vehicles that passed through c to the set $G_c \cup B_c$.

7 ATTACK IMPLEMENTATION FOR THE MINHASH HIERARCHY

In this section, we experimentally validate that our attack on MinHash Hierarchy substantially narrows down the set of possible checkpoints visited by a vehicle to approximately only 10% of all checkpoints in the area.

7.1 Dataset

The dataset used in the original paper [9] is not publicly available. We thus used another public dataset of vehicle location trajectories [24] for the city of Porto, Portugal. Each entry in the dataset defines a vehicle trajectory. The trajectory is described as a list of points, where each point is a pair containing the latitude and longitude of the taxi at a given time point.

7.2 Methodology

For our experiments, we take the first $n = 30\,000$ trajectories in the Porto dataset. As some trajectories contain points that are far outside the city, we removed all points containing an extreme latitude or longitude. We defined a latitude as extreme if it was below 2% or above 98% of all latitudes in these trajectories. We defined a longitude as extreme analogously. We then created a set C of $m = 7744$ checkpoints by fitting an 88×88 square grid on all points in these trajectories.

To generate the MinHash signature for a vehicle, we compute the MinHash signature of a singleton set containing only the identifying number of the vehicle (taken in the $\{1, \ldots, 30\,000\}$ range) using $k = 200$ hash functions. We then computed the checkpoints' signatures from the vehicles passing by, assuming every trajectory belongs to a different vehicle. Each vehicle's GPS coordinates in its trajectory generates one update for the closest checkpoint. Finally, we run our attack on MinHash Hierarchy and for the MinHash z of each vehicle, we compute the sets B_z, G_z, and W_z. We then measure $A_z := |G_z \cup B_z| / |C|$, the ratio of checkpoints that our attack identifies as possibly visited by the vehicle to the total number of checkpoints. The quality of our attack is measured by how low A_z is on average for all vehicles we tested. A_z is around $10\% \pm 5\%$, showing that on average, we narrow down the set of checkpoints visited by the vehicle to only 10% of all checkpoints in the map.

We execute this attack 5 times. Each time, we use a separate set of 30 000 different trajectories.

7.3 Results

Fig. 8 shows a heatmap with 30 000 trajectories. Each pixel is a checkpoint and its brightness is proportional to the number of vehicles that visited that checkpoint.

Figs. 9a and 9c show two example trajectories, using the square grid from Fig. 8. The checkpoints visited by the vehicle are in black. Fig. 9b and Fig. 9d show the outcome of our attack for these two trajectories, respectively. The checkpoints in B_z, G_z, and W_z are

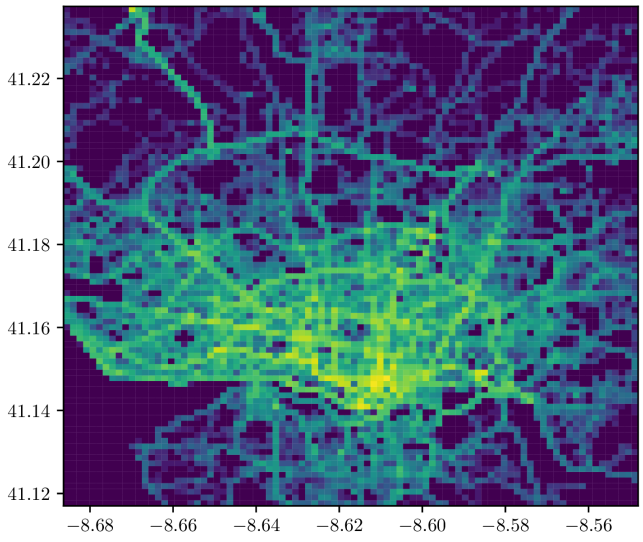

Figure 8: Selected trajectories of checkpoint 2D histogram

marked black, gray, and white, respectively. For the recovery part, we note that trajectory B (28 checkpoints) is hidden within other trajectories (2018 checkpoints). However, some checkpoint signatures had values equal to the vehicle signature (16 checkpoints), and therefore those checkpoints are very likely to be part of the trajectory. For trajectory A, the attack can further isolate the target (27 checkpoints), narrowing it down to only two possible trajectories. More trajectories are included in Fig. 9 for illustration. These trajectories illustrate how, from only the checkpoints' MinHash signature, our attack can either accurately retrieve the target trajectories or restrict it to a much smaller area, thereby compromising the users' privacy.

In our dataset with 30 000 trajectories, a trajectory has on average 25.9±15.6 checkpoints. A set of checkpoints found by our attack has on average 805.5 ± 439.7 checkpoints. Recall that the total number of checkpoint is 7744. Hence on average, we can reduce the set of possible checkpoints visited by a vehicle to around 10% of the original set of checkpoints. This means that in a city like Porto we would restrict the trajectory to a neighborhood. Our attack breaks the MinHash Hierarchy's claimed privacy protection and shows how to confine the target trajectory to a small portion of the map.

Fig. 10 shows the cumulative distribution function (*CDF*) of the size of anonymity sets for the checkpoints. The CDF is computed based on a dataset of 30 000 trajectories and 7744 checkpoints. Each trajectory is treated as a distinct user, and the x-axis denotes the range of possible anonymity set sizes on a logarithmic scale. The y-axis indicates the percentage of checkpoints where users have an anonymity set size less than or equal to x. Similar plots were observed for the remaining trajectories. Our findings reveal that a significant majority of checkpoints are never visited by anyone (approximately 61%). Furthermore, over 1% of the checkpoints have an anonymity set size of 1 (green area), around 5% (green+yellow) have an anonymity size below 30, and 10% (green+yellow+orange) of the checkpoints exhibit an anonymity set size below 270. These observations highlight that, for a small yet substantial fraction

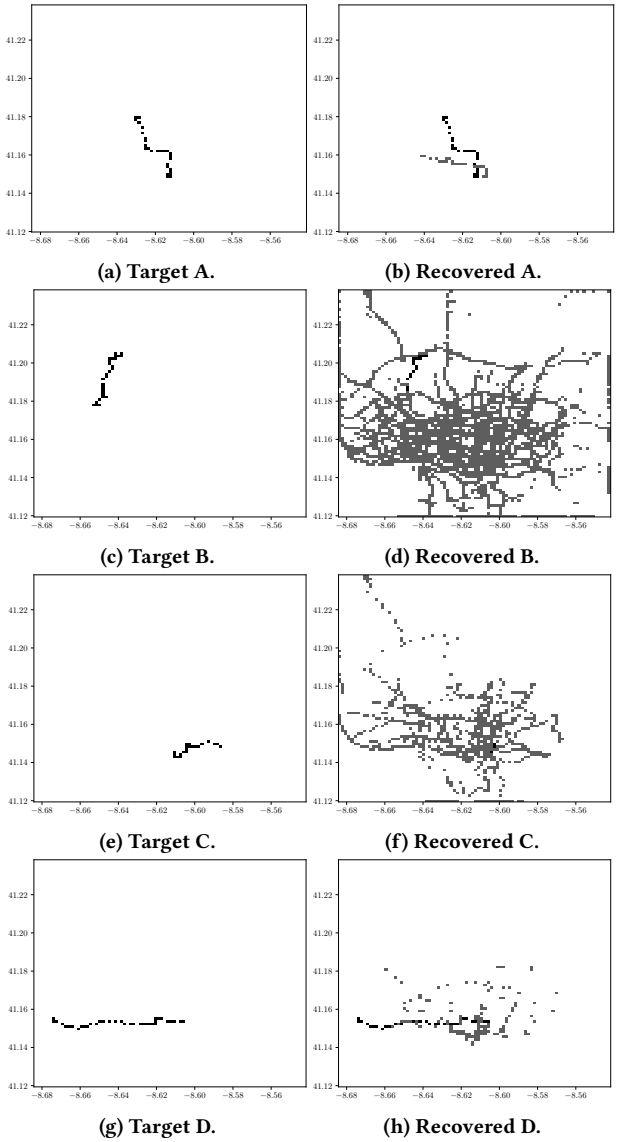

Figure 9: Example of target and recovered trajectories

of users, compromising the privacy of their visited checkpoint locations can be accomplished with relative ease.

7.4 Discussion

We have shown that it is possible to isolate trajectories from checkpoint signatures with good accuracy, restricting a vehicle's potential trajectory to 10% of the checkpoints on average. This breaks the privacy guarantees of MinHash Hierarchy such as differential privacy as claimed by Ding et al.

If a checkpoint c's MinHash signature contains a vehicle's hash, then we are certain (modulo the negligible probability of a hash collision) that the vehicle visited this checkpoint. Each checkpoint signature has 200 hash functions. We can therefore deanonymize

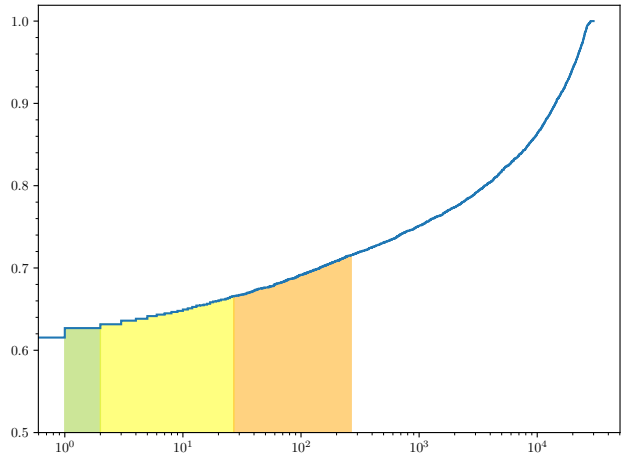

Figure 10: CDF of anonymity set size (x-axis) and the portion of checkpoints with such an anonymity set (y-axis).

up to 200 vehicles. In the city center, due to higher density of people, we can deanonymize a lower fraction of vehicles, while in a rural area we might be able to deanonymize all vehicles. Consequently, discarding trajectories with extreme latitudes or longitudes decreases the effectiveness of our attack, as our attack performs better for rural areas. This preprocessing was necessary to handle the given dataset.

As a post-processing step, it is possible to further filter the set G_z based on time constraints, common commuting patterns, and other external information. We did not explore these techniques since we focus on showing the information leakage stemming from the application of LSH to sensitive data alone. While the MinHash Hierarchy system does not explicitly define temporal properties, it is reasonable to assume that the data collection process is periodically reset, such as on a daily basis, to assess the popularity of trajectories over time. If such an approach is employed, it can potentially amplify the effectiveness of attacks when vehicles consistently select similar routes across multiple iterations, as we have proposed in our previous discussion on FLoC, see Section 4.3. This is particularly relevant for individuals who commute along the same path or similar paths every day, as their likelihood of being included in the MinHash of checkpoints is significantly increased.

8 INSUFFICIENCY OF COUNTERMEASURES

We now discuss some mitigations that have been proposed to address the privacy flaws in LSH systems. We start by arguing that standard solutions like different privacy (*DP*) are insufficient to fix the vulnerabilities of LSH-based systems. This is mainly because LSH-based systems were not designed using DP-mechanisms that take into account how LSH-systems work. Our attacks demonstrate that, in its current state, LSH is not suitable for the design of privacy mechanisms. Novel mechanisms, potentially based on differential privacy, would need to be developed to ensure the privacy properties of these LSH systems.

We now discuss in more detail why standard mitigations do not suffice to counter our attacks.

FLoC. For the FLoC proposal, one option is to make the Chrome operated server, which receives the SimHash and assigns the cohort IDs, trusted and unable to read sensitive information. This can be achieved, for example, by a trusted third party. However, this would prevent neither the Sybil nor the GAN-IP attack. This is because any party can observe a user's cohort ID, which is assigned based on a prefix of the user's SimHash. The Sybil attack can generate Sybil browsing histories that are mapped to that cohort. As the number of such Sybil histories grow, the users' cohort ID would become longer and reveal more of the user's SimHash, which enables the GAN-IP attack.

Recent work [1] proposes using differentially private clustering to compute the cohort IDs. It adds noise to each individual SimHash and then computes from the set Z of all SimHashes a new smaller set Z' of SimHashes, called a *coreset*. Each SimHash z in the coreset acts as a "representative" of a subset S_z of SimHashes in Z that are close to each other. The SimHash z is computed using an additive-noise mechanism and comes with a positive number that approximately indicates the size of S_z. As a result, one cannot infer individual SimHashes in Z from z.

The use of coresets prevents us from conducting the pre-image attack on SimHashes from real individuals, as we cannot retrieve them. However, the SimHashes in the coreset are still vulnerable to the other attacks. In particular, we can still perform the Sybil attack on a SimHash in the coreset so that a SimHash there eventually represents mostly Sybil users. This would then isolate real users and we can then conduct the GAN-IP attack to infer parts of the browsing history of those users.

We argue that the best mitigation is a design of a new system that builds on differential privacy from the start, rather than k-anonymity, to prevent Sybil attacks. This new system should also avoid leaking information in the LSH hashes. The new proposal of the Topics API [18] appears to satisfy both of these requirements but its implementation must still be formalized to allow a thorough evaluation of its privacy guarantees.

MinHash Hierarchy. For the MinHash Hierarchy, the use of differential privacy would provide guarantees about how much information any attack can extract. For example, with a low probability, if a checkpoint would compare its aggregated hash value with a random value instead of the current vehicle hash, it would lower the precision of our attack and give plausible deniability to vehicles.

Recent works propose differentially-private versions of MinHash like PrivRec [37] and PrivMin [36]. However, these proposals provide privacy only for the individual MinHashes and not for systems that process collections of MinHashes, like MinHash Hierarchy. Recall that each checkpoint computes the MinHash of the IDs of mobile devices that pass near the checkpoint. Using PrivMin or PrivRec on each checkpoint provides differential privacy, but only for an *individual* checkpoint. One must still demonstrate that PrivMin and PrivRec's DP guarantees can tolerate the computations that MinHash Hierarchy conducts using the MinHashes from multiple checkpoints.

In summary, systems that process sensitive information from users must protect their privacy. Current systems based on LSH can provide stronger guarantees if they are enhanced with appropriate differential-privacy mechanisms, but the state of the art in differential privacy is still unable to provide this. Our work demonstrates

the need for novel solutions that provide better privacy protections for LSH-based systems.

9 RELATED WORK

Attacks on FLoC. Berke et al. [4] emulated the FLoC algorithm producing cohorts over time, using a proprietary (paid) demographic and browsing history dataset. They then attacked the algorithm using the uniqueness of browsing histories over time and tracking sequences of FLoC IDs. They could identify 95% of the users after 4 weeks. Combining this attack with standard fingerprinting techniques would make it even more effective. In addition, with the observed data, they could connect users' racial backgrounds to their browsing histories, in spite of the fact that they found no direct connection between race and cohorts. Berke et al.'s work focuses on the tracking of individual users and the correlation between cohorts and sensitive demographics. They show that FLoC enables the tracking of individual users, which is an alternative to our Sybil attack. However, they do not reconstruct users' browsing histories as our GAN-IP attack does. Furthermore, our attack also works without the need of collecting data over a long period, which is a requirement for the attack of Berke et al.

Mozilla also mentioned in their report [28] that FLoC was vulnerable to Sybil attacks. However, their claims were neither formally verified nor experimentally validated. Given that FLoC was only tested during a trial with limited user participation, the majority of attacks remained theoretical with no practical implementation. Our work not only gives a theoretical analysis, but also provides a practical implementation and an experimental evaluation using real datasets.

Attacks on Perceptual Hashing and NeuralHash. Another type of LSH is *perceptual hash*, which is used for images. A perceptual hash is a fingerprint computed from an input image. It is possible to mount pre-image attacks using conditional adversarial GANs (cGANs), like Pix2Pix [16]. Such GANs learn how to translate images in one style to another (e.g., translating a hand-drawn sketch of a bag to a photo of a bag). The attack trains a cGAN that learns to translate perceptual hashes to possible pre-images with a success rate of 30% [20]. This makes perceptual hash unsuitable for privacy applications. It remains as future work to determine whether cGANs would be successful in mounting pre-image attacks on FLoC or MinHash.

Another instance of a perceptual hashing function is NeuralHash, used by Apple for Child Sexual Abuse Material (*CSAM*) Detection [2]. To detect abusive images, their system stores hashes computed using convolutional neural networks (CNN) and LSH. Their model is vulnerable to adversarial attacks [3, 19] that can lead to non-abusive images being labeled as abusive. To our knowledge, NeuralHash has not been shown to be resistant to pre-image attacks. It remains as future work to investigate what private information the hash reveals about an image.

Criticisms of FLoC. While we are the first who implemented and evaluated FLoC's privacy leakage, we were not the first to criticize it. Other browser vendors pointed out FLoC's potential privacy issues (Mozilla [28], Brave [31], Vivaldi [35]) as well as NGOs (e.g., EFF [8]) and advertisers [30]. For example, FLoC further strengthens already existing hard-to-counter fingerprinting schemes and cohort IDs can be used to further partition users according to browsing behaviors, making tracking easier. Also, FLoC requires a trusted Chrome server that ensures k-anonymity and removes sensitive cohorts. However, to the best of our knowledge, a server fulfilling this requirement has not been presented yet. The Chrome server would allow Google to centralize the collection of SimHashes, creating a conflict of interest for Google. This would also strengthen Google's monopoly on advertising and tracking.

10 CONCLUSIONS

In this work, we studied two systems that use locality sensitive hashing (LSH) to privately handle user data. The designers of both systems considered LSH to be privacy preserving, and, in both cases, we showed how to reconstruct a significant portion of the private inputs from just the hashes. Namely, for MinHash Hierarchy, we extracted parts of vehicle trajectories that were intended to be hidden by the MinHashes computed by the checkpoints. For Google's FLoC, we could construct pre-images that enable an efficient Sybil attack, and from the hashes we reconstructed parts of browsing history. Although Google discontinued FLoC, they had tested it on tens of millions of users, underscoring their serious interest in using LSH. Our findings, together with other observed attacks like Apple's Child Sexual Abuse Material Detection, show that LSH hashes leak substantial information about private data, a fact that has been systematically overlooked.

Our findings show the importance of evaluating the privacy leakage of any system handling sensitive data. We leave for future work the study of other systems, such as the Topics API, systems that use perceptual hashing, and systems that use differential privacy without a proper evaluation of the information leakage under multiple queries.

DATA AVAILABILITY

Further information and updates of this publication are available at https://karelkubicek.github.io/post/floc. Our attacks' implementations are available at https://github.com/privacy-lsh/floc-minhash. For further details on the FLoC attack, we refer to our technical report [34]. The datasets used to evaluate our work are available from the corresponding publications, namely Porto Taxi dataset by Moreira et al. [24] and MovieLens by Harper et al. [14].

ACKNOWLEDGMENTS

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

We thank Hung Hoang for his advice on integer programming and Matteo Scarlata for his valuable feedback. We also thank the MinHash Hierarchy authors for providing us with the implementation of the MinHash signature computation in their system.

REFERENCES

[1] Google Cloud Alisa Chang and Google Research Pritish Kamath. 2021. Practical Differentially Private Clustering. https://ai.googleblog.com/2021/10/practical-differentially-private.html.
[2] Apple. 2021. CSAM Detection Technical Summary. https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf.
[3] Anish Athalye. 2021. NeuralHash Collider. https://github.com/anishathalye/neural-hash-collider.

[4] Alex Berke and Dan Calacci. 2022. Privacy Limitations Of Interest-based Advertising On The Web: A Post-mortem Empirical Analysis Of Google's FLoC. arXiv:2201.13402 [cs.CY]

[5] Dumitru Brinza, Matthew Schultz, Glenn Tesler, and Vineet Bafna. 2010. RAPID detection of gene–gene interactions in genome-wide association studies. *Bioinformatics* 26, 22 (2010), 2856–2862.

[6] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, IEEE Computer Society, 1730 Massachusetts Ave., NW Washington, DC, United States, 21–29.

[7] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. Association for Computing Machinery, New York, NY, United States, 380–388.

[8] Bennett Cyphers. 2021. Google's FLoC Is a Terrible Idea. https://www.eff.org/deeplinks/2021/03/googles-floc-terrible-idea.

[9] Jiaxin Ding, Chien-Chun Ni, Mengyu Zhou, and Jie Gao. 2017. MinHash Hierarchy for Privacy Preserving Trajectory Sensing and Query. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. Association for Computing Machinery, New York, NY, United States, 17–28.

[10] John R Douceur. 2002. The Sybil attack. In *International workshop on peer-to-peer systems*. Springer, Springer Berlin, Heidelberg, Cambridge, MA, USA, 251–260.

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, Springer International Publishing, New York, NY, USA, 265–284.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[13] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2017. Long Text Generation via Adversarial Training with Leaked Information. arXiv:1709.08624 [cs.CL]

[14] F Maxwell Harper and Joseph A Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (Dec. 2015), 19:1–19:19. https://doi.org/10.1145/2827872

[15] Taher Haveliwala, Aristides Gionis, and Piotr Indyk. 2000. Scalable techniques for clustering the web. *WebDB Workshop* 129 (2000), 134.

[16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2018. Image-to-Image Translation with Conditional Adversarial Networks. arXiv:1611.07004 [cs.CV]

[17] Yushi Jing and Shumeet Baluja. 2008. VisualRank: Applying PageRank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 11 (2008), 1877–1890.

[18] Michael Kleber Josh Karlin. 2022. Topics API GitHub. https://github.com/patcg-individual-drafts/topics.

[19] Lim Swee Kiat. 2021. Apple NeuralHash Attack. https://github.com/greentfrapp/apple-neuralhash-attack.

[20] Nick Locascio. 2018. Black-Box Attacks on Perceptual Image Hashes with GANs. https://towardsdatascience.com/black-box-attacks-on-perceptual-image-hashes-with-gans-cc1be11f277.

[21] Mohammad Malekzadeh, Richard G. Clegg, and Hamed Haddadi. 2018. Replacement AutoEncoder: A Privacy-Preserving Algorithm for Sensory Data Analysis. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, Orlando, FL, USA, 165–176. https://doi.org/10.1109/iotdi.2018.00025

[22] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*. Association for Computing Machinery, New York, NY, USA, 141–150.

[23] Don Marti. 2021. Early Status of the FLoC Origin Trials. https://cafemedia.com/early-status-of-the-floc-origin-trials/.

[24] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. 2013. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems* 14, 3 (2013), 1393–1402.

[25] The Chromium Projects. 2021. FLoC Origin Trial & Clustering. https://www.chromium.org/Home/chromium-privacy/privacy-sandbox/floc.

[26] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of massive datasets*. Cambridge University Press, Shaftesbury Road, Cambridge, UK.

[27] Deepak Ravichandran and Sergei Vassilvitskii. 2021. Evaluation of Cohort Algorithms for the FLoC API. https://github.com/google/ads-privacy/blob/master/proposals/FLoC/FLOC-Whitepaper-Google.pdf.

[28] Eric Rescorla and Martin Thomson. 2021. Technical Comments on FLoC Privacy. https://mozilla.github.io/ppa-docs/floc_report.pdf

[29] Antoine Rouzaud. 2021. FLoC Origin Trial Observations. https://medium.com/@antoine.rouzaud.

[30] Allison Schiff. 2021. The Industry Reacts To Google's Bold Claim That FLoCs Are 95% As Effective As Cookies. https://www.adexchanger.com/online-advertising/the-industry-reacts-to-googles-bold-claim-that-flocs-are-95-as-effective-as-cookies/.

[31] Peter Snyder and Brendan Eich. 2021. Why Brave Disables FLoC. https://brave.com/why-brave-disables-floc/.

[32] Benno Stein. 2007. Principles of hash-based text retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. Association for Computing Machinery, New York, NY, USA, 527–534.

[33] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems* 10, 05 (2002), 557–570.

[34] Florian Turati. 2022. *Analysing and exploiting Google's FLoC advertising proposal*. Master's thesis. ETH Zurich, Department of Computer Science.

[35] Jon von Tetzchner. 2021. No, Google! Vivaldi users will not get FLoC'ed. https://vivaldi.com/blog/no-google-vivaldi-users-will-not-get-floced/.

[36] Ziqi Yan, Jiqiang Liu, Gang Li, Zhen Han, and Shuo Qiu. 2017. PrivMin: Differentially Private MinHash for Jaccard Similarity Computation. https://doi.org/10.48550/ARXIV.1705.07258

[37] Yifei Zhang, Neng Gao, Junsha Chen, Chenyang Tu, and Jiong Wang. 2020. PrivRec: user-centric differentially private collaborative filtering using LSH and KD. In *International Conference on Neural Information Processing*. Springer, Springer International Publishing, New York, NY, USA, 113–121.

Table 8: Example for the GAN-IP attack. We highlight intersections of target and generated histories .

Target history h from test data:	**Generated history h':**	**Subset of movies selected by int. prog.:**
American President, The (1995)	Ace Ventura: Pet Detective (1994)	Ace Ventura: Pet Detective (1994)
Birdcage, The (1996)	Aladdin (1992)	Batman (1989)
Client, The (1994)	Batman (1989)	Beauty and the Beast (1991)
Crimson Tide (1995)	Beauty and the Beast (1991)	Braveheart (1995)
Dances with Wolves (1990)	Braveheart (1995)	Clear and Present Danger (1994)
Dead Man Walking (1995)	Clear and Present Danger (1994)	Cliffhanger (1993)
Die Hard: With a Vengeance (1995)	Cliffhanger (1993)	Crimson Tide (1995)
Disclosure (1994)	Crimson Tide (1995)	Disclosure (1994)
English Patient, The (1996)	Die Hard: With a Vengeance (1995)	Firm, The (1993)
Fargo (1996)	Disclosure (1994)	Jurassic Park (1993)
Firm, The (1993)	Firm, The (1993)	Lion King, The (1994)
Forget Paris (1995)	GoldenEye (1995)	Outbreak (1995)
Grumpier Old Men (1995)	Jurassic Park (1993)	Pulp Fiction (1994)
Lion King, The (1994)	Lion King, The (1994)	Seven (a.k.a. Se7en) (1995)
Mirror Has Two Faces, The (1996)	Outbreak (1995)	Shawshank Redemption, The (1994)
Mission: Impossible (1996)	Pulp Fiction (1994)	Silence of the Lambs, The (1991)
Mrs. Doubtfire (1993)	Seven (a.k.a. Se7en) (1995)	Star Trek: Generations (1994)
Mr. Holland's Opus (1995)	Shawshank Redemption, The (1994)	True Lies (1994)
Nell (1994)	Silence of the Lambs, The (1991)	Twelve Monkeys (1995)
Outbreak (1995)	Star Trek: Generations (1994)	Twister (1996)
Philadelphia (1993)	True Lies (1994)	While You Were Sleeping (1995)
Postman, The (Postino, Il) (1994)	Twelve Monkeys (1995)	
Rock, The (1996)	Twister (1996)	
Sabrina (1995)	While You Were Sleeping (1995)	
Seven (a.k.a. Se7en) (1995)		
Shawshank Redemption, The (1994)		
Silence of the Lambs, The (1991)		
Spy Hard (1996)		
Sudden Death (1995)		
Toy Story (1995)		
Twelve Monkeys (1995)		
Twister (1996)		