The Pitfalls of Hashing for Privacy

Levent Demir, Amrit Kumar, Mathieu Cunche, Cédric Lauradoux

The Pitfalls of Hashing for Privacy

(This is an author version of the article published in IEEE Communications Surveys & Tutorials (Volume 20, Issue 1, 2018))

Levent Demir, Amrit Kumar, Mathieu Cunche and Cédric Lauradoux

***Abstract*—Boosted by recent legislations, data anonymization is fast becoming a norm. However, as of yet no generic solution has been found to safely release data. As a consequence, data custodians often resort to ad-hoc means to anonymize datasets. Both past and current practices indicate that hashing is often believed to be an effective way to anonymize data. Unfortunately, in practice it is only rarely effective. This paper is a tutorial to explain the limits of cryptographic hash functions as an anonymization technique. Anonymity set is the best privacy model that can be achieved by hash functions. However, this model has several shortcomings. We provide three case studies to illustrate how hashing only yields a weakly anonymized data. The case studies include MAC and email address anonymization as well as the analysis of Google Safe Browsing.**

***Index Terms*—Anonymity set, Anonymization, Balls-into-bins, Hashing, Pseudonymization.**

I. Introduction

Personally identifiable information, or in general, data with social or economic value are collected by almost all service providers. The data help a service provider identify, contact or locate discrete individuals, which in turn may be necessary to deliver the intended service. The personally identifiable information handled by these service providers can however be used to the detriment of an individual's privacy. Much of the success of these services hence depends on the ability to ensure that the privacy of each individual is respected. One of the most effective ways to do so is *data anonymization*.

In simple terms, data anonymization consists in processing personally identifiable data in order to prevent identification, hence, mitigating the privacy risks for the concerned individuals. Data anonymization can be applied either when the data is collected, communicated, or stored or when the data is published in the public domain for further analysis or scrutiny.

With privacy being an increasing concern, data anonymization is rapidly becoming a norm rather than an exception. This particularly owes to the different legislations and directives that have been enacted in the recent past [1]–[5]. Considering the trade-off between utility and convenience, companies and administrators often turn to *pseudonymization* instead of full scale anonymization [6]–[9].

Pseudonymization can be seen as a way to hide the real identity of an individual (or an identifier such as his social security number) by replacing it with a false one so that information related to the individual can be handled (as usual) without knowing the real identity. It reduces the linkability of a dataset with the original identity of a data subject and hence mitigates privacy risks.

A popular choice to pseudonymize a data is through hashing [6]–[9]. More precisely, cryptographic hash functions are often used to pseudonymize identifiers as these functions are *one way*, *i.e.*, irreversible. This protects the identities of the individuals, but only ostensibly so. Indeed, while it may appear to be a good privacy rationale, the final results of the "*hashing for anonymization*" approach have been catastrophic.

In fact, re-identification attacks have been made possible on the published datasets and consequently sensitive data are now available in the wild [6], [10], [11]. There are three main reasons for this failure. First, the *one-wayness* of hash functions is misunderstood and data custodians often underestimate the risk of exhaustive search. Second, even when exhaustive search cannot be carried out on the initial domain space, it should however be possible to do so on one of its subsets. This may allow an attacker to learn a discriminating property (associated to the subset) about a pseudonymized data. Finally, the following privacy argument is quite prevalent among data custodians and software developers: Hashing cannot take into account any prior adversary knowledge. The argument clearly undermines the adversarial strength.

Hashing of unique identifiers is being used by the companies and by government agencies to provide some privacy protection to users. Hashing for anonymizing MAC addresses [8], [12] has even been adopted by Wi-Fi tracking industry as evidenced by its inclusion in the Mobile Location Analytics Code of Conduct [13]. The German federal agency [14] also used to anonymize and distribute sensitive list of URLs. However, as noted by several third parties [15]–[17], this method does not really protect privacy as it can be trivially reversed. More recently, the French data protection agencies rejected [18] the deployment of a Wi-Fi tracking system in Paris using hashing to anonymize MAC addresses. This practices are not limited to the industry, as researchers also have used it to create and distribute datasets such as computer network traces [19], [20].

This paper is a tutorial that explains the difficulty to (pseudo)anonymize data with hash functions. The tutorial is self-contained and hence does not require any prior background in the field of privacy. Readers are however assumed to have some basic notions of discrete probability.

The tutorial presents compelling evidences that put into light some of the mistakes made in the past when data custodians and administrators have employed hashing for (pseudo)anonymization. The motivations for this work is to improve data anonymization in practice by presenting well founded arguments to dissuade data custodians and administrators from using weak (pseudo)anonymization schemes.

The tutorial is divided into two parts. The first part es-

L. Demir is attached with INCAS-ITSec and Inria, France. Work done while A. Kumar was attached with Université Grenoble Alpes, France. C. Lauradoux and M. Cunche are attached with Inria, France.

e-mails: levent.demir@inria.fr; amrit.kumar@m4x.org; mathieu.cunche@inria.fr; cedric.lauradoux@inria.fr

tablishes the privacy argument that can be used when hash functions are used for anonymization. To this end, we study *anonymity set* and propose to estimate its size using the probabilistic model of *balls-into-bins*.

The second part focuses on three case studies namely Gravatar — a globally recognized avatar, Wi-Fi tracking systems based on MAC addresses and GOOGLE Safe Browsing — a malicious URL detection tool. The Gravatar case study shows how a basic usage of hashing fails to protect email addresses. As for MAC addresses, we first show that the basic usage of hashing to anonymize them fails for the same reason it fails in the case of Gravatars. We also explain why the anonymity set argument would fail too. To conclude, we analyze GOOGLE Safe Browsing, where a hash function is used to create pseudonyms for malicious URLs. Our analysis studies the associated anonymity set and its limits.

We note that re-identification in case of Gravatars has been previously studied by Bongard [10]; the same for MAC addresses by Demir *et al.* [21], while a privacy analysis of Safe Browsing was recently conducted by Gerbet *et al.* [22]. This tutorial systematizes these independent studies using a uniform framework to highlight the pitfalls of hashing for privacy.

The tutorial is organized as follows. Section II provides the basic definitions on hash functions. The pseudonymization setting and adversarial goals are described in Section III. In the following two sections: Section IV-A and Section IV-B, we study these adversaries under different hypotheses on the underlying hash function. Section IV-A considers the case when the hash function behaves as a one-to-one map between the set of identifiers and the set of pseudonyms, while in Section IV-B, we study the case when the hash function only behaves as a many-to-one map. While, the former represents the case of pseudonymization, the latter models into the anonymity set like privacy argument. Building upon these theoretical considerations, we discuss three case studies: Gravatar (Section V), MAC addresses (Section VI) and GOOGLE Safe Browsing (Section VII).

II. BACKGROUND ON HASH FUNCTIONS

A. Definitions

A hash function h compresses inputs of arbitrary length to a *digest/hash* of fixed size: $h : \{0,1\}^* \to \{0,1\}^\ell$. Hash functions are assumed to be *uniform*, which means that given a random input from $\{0,1\}^*$, each digest in $\{0,1\}^\ell$ is equally likely.

There exist two families of hash functions: *collision-resistant hash functions* and *one-way hash functions* [23]. These two families are often not mutually exclusive in case of hash functions designed for cryptographic use. Below, we discuss these notions.

A collision for a hash function is a set of two inputs x_1 and x_2 such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$. As the size of inputs is arbitrary, while the size of outputs is fixed, due to the pigeonhole principle there exist collisions for h. However, if h is a cryptographic hash function, it must be difficult to find such collisions (see [23] for further detail). The notion of a collision can be naturally extended to *r-collisions*: a set of r inputs $x_1, x_2, \ldots, x_r$ such that $x_1 \neq x_2 \neq \cdots \neq x_r$ and $h(x_1) = h(x_2) = \cdots = h(x_r)$.

One-wayness is another property particularly important in the setting of anonymization. A one-way hash function guarantees the following two properties:

- **Pre-image resistance:** Given a digest d, it is computationally infeasible to find any input x, such that $h(x) = d$.
- **Second pre-image resistance:** Given an input x and the digest $h(x)$, it is computationally infeasible to find another input $x' \neq x$, such that $h(x) = h(x')$.

The NIST recommendation [24] for cryptographic hash functions are SHA-256, SHA-384, SHA-512 [25] and SHA-3 [26] with $\ell \geq 256$.

Let us consider a one-way hash function h which outputs ℓ-bit digests. The choice of ℓ is critical for h because the basic complexities for finding pre-images, second pre-images and collisions are 2^ℓ, 2^ℓ and $2^{\ell/2}$ respectively. The first two complexities correspond to brute force attacks — the best generic attack that an attacker can mount. The complexity of finding a collision is a consequence of the *birthday paradox* detailed below. The digest size of the main cryptographic hash functions is given in Table I.

TABLE I: Digest size and status of the prominent cryptographic hash function the

Hash function	Digest size ℓ (in bits)	Status
MD4	128	Unsafe
MD5	128	Unsafe
SHA-1	160	Unsafe
SHA-256	256	Safe
SHA-512	512	Safe
SHA-3	224 to 512	Safe

B. Birthday Paradox

Throughout this tutorial, we will be using a result from probability that has an important impact on the collision resistance of a hash function. The result is often referred to as the *birthday paradox* and leads to a well-known cryptographic attack called the *birthday attack*.

The mathematical setting for the paradox can be described in the following manner: We throw m balls into n bins sequentially by placing each ball into a bin chosen independently and uniformly at random. Now, the question is to compute the probability to have a bin with at least two balls into it. When rephrased in terms of birthdays, m may represent the number of randomly chosen people in a group (balls), and n the number of days in a year (bins). The required probability then refers to the chance that two people in the group share the same birthday.

It is also easy to see that the setting directly relates to the probability of finding a collision in a hash function. In fact, a hash function can be used to model the assignment of a ball to a bin. To this end, m may represent the number of messages (balls) to be hashed, and n the total number of possible digests (bins). The required probability then refers to the chance that one is able to find a collision among the m messages.

We note that when $m > n$, by the pigeonhole principle, the probability that there exists a bin with at least two balls into it is 1. We hence restrict to a more interesting scenario where $m \leq n$.

One can show that the number of balls required to generate a collision with probability p is given by:

$$m \approx \sqrt{2n \cdot \ln\left(\frac{1}{1-p}\right)} \ .$$

Interested readers may refer to [23] for further details.

As a consequence, to obtain a collision probability of 40%, it suffices to have $m = \sqrt{n}$. Translating the result in the terminology of hash functions which generate ℓ-bit digests, the expected number of random samples that should be generated before getting a collision (with reasonably high probability) is not 2^ℓ, but rather only $2^{\ell/2}$ (roughly).

The birthday attack to find a collision for a hash function [27] consists in two steps:

1) Generate values x and their respective digests $H(x)$
2) Insert $(x, H(x))$ in a hash table and check if $H(x)$ is already in the table.

C. Construction

Several cryptographic hash functions such as MD5, SHA-1 and SHA-2 family are constructed using an iterative procedure referred to as the Merkle-Damgård construction [28]–[30]. Below, we give an overview of this construction paradigm.

The underlying idea of the Merkle-Damgård paradigm is the following: Instead of directly constructing a hash function that operates on inputs of arbitrary size, construct a hash function in an iterative manner from a *compression function* that operates on a fixed-sized input. More precisely, a compression function f_c is defined as $f_c : \{0,1\}^p \times \{0,1\}^t \to \{0,1\}^t$. It takes an input of a fixed size p, an internal state of size t and compresses the input to t bits. In fact, any hash function that treats the input message as a stream can be seen as an iterative application of some compression function.

The simplest way to construct an iterative hash function is to use the last computed value as the final digest. Consider a message block $m = (m_1, m_2, \ldots, m_k)$, where each $m_i \in \{0,1\}^p$ and the initial state $s_0 = IV \in \{0,1\}^t$. IV is a constant value referred to as the *initialization vector*. The output at the i-th iteration is then given by $s_i = f_c(s_{i-1}, m_i)$. The final digest is s_k. See Figure 1 for a schematic representation. This construction yields a hash function h such that $h(m) = s_k$.

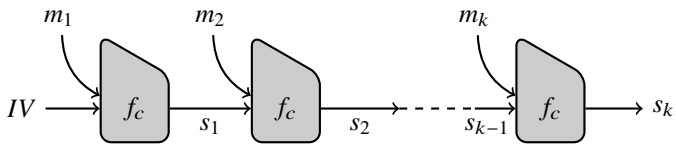

Fig. 1: Merkle Damgård construction of an iterative hash function.

The above construction allows to hash an arbitrary long message, but does not handle message blocks whose size is not a multiple of p. In order to handle messages of all sizes, the construction requires a preprocessing using a *padding* function. Padding takes an input message and returns an output whose length is a multiple of p. The padding function has to be injective as a collision in the padding function would lead to a collision in the hashing algorithm. The simplest padding consists in: 1) First, adding a bit `1` followed by a sequence of bit `0` (between 0 and $p-1$) in order to have a size that is a multiple of p, 2) Second, encoding the message length as a 64-bit integer.

The choice of the hash functions is irrelevant as long as the function is considered cryptographically secure. The current recommendation of the NIST is SHA-3. The status of many cryptographic hash functions can be found in [31], [32]. In several of our case studies, the hash functions SHA-1 (Section VI) or MD5 (Section V) were used. They were still considered secure at the time they were used.

III. Settings and Adversaries

A. Settings

Let us assume that we want to create pseudonyms for a finite set $\mathcal{A}$ of m identifiers $\mathrm{id}_1, \ldots, \mathrm{id}_m$ using a one-way hash function. We obtain a set $\mathcal{P}$ of pseudonyms $h(\mathrm{id}_1), \ldots, h(\mathrm{id}_m)$ such than $|\mathcal{P}| \leq m$. We note that an inequality rather than an equality is used here because collisions can occur and hence pseudonyms may not necessarily be all distinct. For the rest of the discussion, we denote the size of $\mathcal{P}$ by n.

It is important to distinguish two scenarios depending on the number of identifiers associated to a pseudonym. In the first case, at most one identifier is associated to a pseudonym. This one-to-one mapping ($m = n$) implies that the hash function is injective. This case occurs when $m \ll 2^{\ell/2}$ (Birthday paradox bound). Figure 2 schematically presents this case.

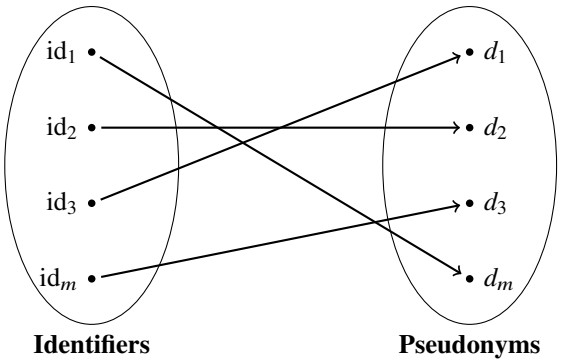

Fig. 2: A one-to-one pseudonymization. The pseudonym $d_i = h(\mathrm{id}_i)$, where h is the underlying hash function.

The second case considers the scenario where multiple identifiers get associated to the same pseudonym. It occurs when $m \gg 2^\ell$. This implies that the hash function is not injective, *i.e.*, the mapping is *many-to-one*. Figure 3 schematically presents this case.

B. Adversaries

Pseudonyms so generated can be used in a database or in a protocol. Let us assume that the adversary has gained access to the anonymized database or to the protocol transcript. From the knowledge of a pseudonym d, we need to establish the goals for an adversary. She can either achieve *re-identifcation* (full recovery) of the identifier or attempt to mount a *discrimination*

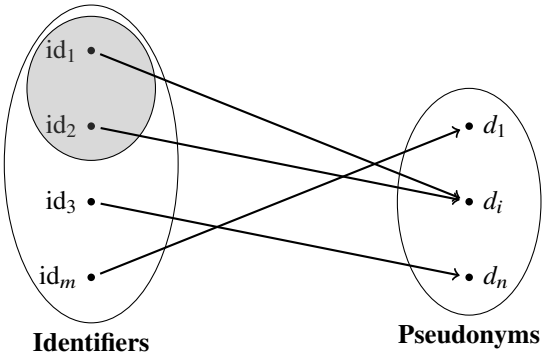

Fig. 3: A many-to-one pseudonymization. id_1 and id_2 both map to the same pseudonym d_i. The pseudonym d_i is generated using the underlying hash function h.

attack. These are the most common and standard attacks against any anonymization scheme (see [1]).

- The re-identification attack is the strongest attack that may be mounted on an anonymized identifier. It implies that the adversary is able to invert the anonymization function.
- The discrimination attack consists for the adversary to determine if a pseudonym belongs to a certain group or to another. The attack assumes that the domain $\mathcal{A}$ can be split into two: If an identifier verifies a certain discriminating property it belongs to $\mathcal{A}_1$, otherwise it belongs to $\mathcal{A}_2$. Clearly, we have $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ and $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$.
  An illustrative scenario of the discrimination attack is to assume that the attacker has a list of identifiers, and wants to know if they have been included in a pseudonymized dataset.

IV. Privacy Provisions of Hash functions

In this section, we study the anonymity achieved when a hash function is used to create pseudonyms. The level of anonymity achieved depends on whether the hash function behaves as a one-to-one mapping on the set of identifiers or is a many-to-one mapping. This behavior will have a direct impact on the success of attacks that an adversary can mount.

A. One-to-One Mapping

The re-identification attack is possible in this case if $\mathcal{A}$ can be enumerated by the adversary in reasonable time. She can compute:

$$h(\mathrm{id}_1), \ldots, h(\mathrm{id}_m),$$

until she finds $h(\mathrm{id}_i) = d$. Since there is no collision, she recovers the unique identifier that corresponds to the pseudonym d. The complexity of this attack is $O(m)$.

The discrimination attack is a re-identification attack on an enumerable (in reasonable time) subset of identifiers. The adversary computes the pseudonyms $h(\mathrm{id}_i), \ldots, h(\mathrm{id}_j)$ associated to the subset $\mathcal{A}_1$ which is assumed to be enumerable in reasonable time. If any of the identifiers in $\mathcal{A}_1$ generates d as its pseudonym, then one can learn that d corresponds to an identifier that verifies the discriminating property associated to $\mathcal{A}_1$. Otherwise, d verifies the property associated to $\mathcal{A}_2$. In the former case, the adversary also learns the unique identifier, while in the latter, she only learns that the identifier does not verify the discriminating property. The complexity of the discrimination attack is $O(|\mathcal{A}_1|)$.

The attacks are feasible as long as the number of identifiers m is small compared to the number of possible digests, *i.e.*, we have a unique mapping between the pseudonyms and the identifiers. If m is small, an exhaustive search enables the re-identification attack. Otherwise, discrimination attacks are still possible even without any prior knowledge or auxiliary information. It clearly jeopardizes the use of hash functions to create pseudonyms. These issues are particularly detailed in Section VI.

B. Many-to-One Mapping

In this scenario, we consider a different privacy model in which several identifiers are associated to the same pseudonym. Hence, the risk of de-anonymizing an identifier is reduced. This model known as *anonymity set* was posited by Pfitzmann and Köhntopp [33]: *"Anonymity is the state of being not identifiable within a set of subjects, the anonymity set"*.

Using a hash function h, a pseudonym d and for a given size r, the *anonymity set* of d is composed of r distinct identifiers $\mathrm{id}_1, \mathrm{id}_2, \ldots, \mathrm{id}_r$ that map to the same pseudonym d, *i.e.*,

$$h(\mathrm{id}_1) = h(\mathrm{id}_2) = \cdots = h(\mathrm{id}_r) = d. \tag{1}$$

Hence, the pseudonym d is anonymous among these r identifiers. In other words, the *anonymity set size* of d is r. The larger the value of r is, the better is the anonymity achieved. We note that Equation 1 matches the definition of r-collisions from Section II.

For a pseudonym to have an anonymity set size of $r > 1$, one must have $m \gg 2^\ell$. In general, this is unlikely to happen if one works with the full digest of a cryptographic hash function. In fact, the digest size needs to be reduced to force collisions. In the rest of this tutorial, we assume without loss of generality that this is achieved by ignoring the least significant bits of the digest.

We note that when considering n distinct anonymity sets of respective sizes $r_1, r_2, \ldots, r_n$, the overall anonymity achieved is given by the smallest size $\min_{1 \le i \le n} r_i$. This is in line with the argument that an anonymization technique for a set of identifiers gives only as much anonymity as its least anonymized identifier. The computation of this value and the required size of the reduced digests can be obtained using the *balls-into-bins* argument [34].

As discussed earlier, the balls-into-bins model consists in throwing m balls into n bins sequentially by placing each ball into a bin chosen independently and uniformly at random. In the current context, identifiers represent the balls, while pseudonyms represent the bins. Hence, m is the total number of identifiers, *i.e.*, $|\mathcal{A}| = m$ and n is the total number of possible pseudonyms, *i.e.*, $|\mathcal{P}| = n$. We also note that the procedure to place a ball into a bin is simulated by the hash function used to generate a pseudonym from an identifier.

In order to measure the overall anonymity, we wish to determine the smallest load for a bin, *i.e.*, $\min_{1 \le i \le n} r_i$. To this end, let us first count the number of ways to throw m balls into n bins such that each bin has at least k balls into it. Since, each of the bins must have at least k balls, we can remove k balls from each bin. After removing these nk balls, we are left with the task of distributing $m - nk$ balls into n bins. The number of ways of distributing $(m - nk)$ balls into n bins is,

$$\binom{m - nk + n - 1}{n - 1} \tag{2}$$

The above term is obtained using the *stars and bars* method from combinatorics due to Feller [35].

Hence, the probability that the minimum load of a bin is *at least* k can be given as:

$$\frac{\binom{m-nk+n-1}{n-1}}{n^m} \tag{3}$$

Finally, the probability that the minimum load of a bin *is exactly* k is given by:

$$\frac{\binom{m-nk+n-1}{n-1} - \binom{m-nk-1}{n-1}}{n^m}, \tag{4}$$

where the second binomial in the numerator is the number of ways to throw m balls into n bins such that every bin has at least $k + 1$ balls. This is obtained from (2) by substituting k by $k + 1$.

For large values of m and n, the general formula given by (4) can turn out to be difficult to compute. To this end, Ercal-Ozkaya [36] proves the following asymptotic result when $m > n \log n$.

*Theorem 1 (*Ercal-Ozkaya [36]*):* For a constant $c > 1$, if one throws m balls into n bins where $m \ge cn \log n$ uniformly at random, then with high probability, the minimum number of balls in any bin is $\Theta\left(\frac{m}{n}\right)$.

Now, we compute the average anonymity achieved for a set of n distinct anonymity sets of respective sizes $r_1, r_2, \ldots, r_n$, *i.e.*, compute the mean value of these r_is. Using the terminology of balls and bins, this is equivalent to computing the average load of a bin. To this end, let X_i be the random variable which counts the number of balls in the bin i. Clearly,

$$\sum_{i=1}^{n} X_i = m.$$

Therefore, $\mathbb{E}[\sum X_i] = m$ and using the linearity of expectation, $\mathbb{E}[X_i] = \frac{m}{n}$. Hence, if $m = n$, we expect to see one ball in each bin. In terms of anonymity set size, this corresponds to an anonymity set size of 1, which implies that on an average there is no collision among the pseudonyms. This case degenerates into the case of one-to-one mapping. Hence, if the set of identifiers $\mathcal{A}$ were enumerable, then it should be possible to mount the re-identification attack. The discrimination attack may also succeed as long as a subset of $\mathcal{A}$ remains enumerable.

A critical hypothesis for anonymity set size is to assume that all the identifiers which form an r-collision have the same probability of appearance. If this condition does not hold, the anonymity set size gets reduced and re-identification and discrimination attacks become plausible. This criticism was made by Serjantov and Danezis in [37].

In the following sections, we apply the above framework to study the level of anonymity achieved in three real-world applications. In Section V, we study the case of a large value of m in the context of email addresses. Despite a large m, this case in effect constitutes a one-to-one pseudonymization. In Section VI, we show how anonymization of MAC addresses fails as the domain space can be reduced to mount discrimination attacks in certain Wi-Fi tracking systems. This case study natively encompasses both one-to-one and many-to-one mappings. Lastly, we discuss the most intricate example of Safe Browsing in Section VII where a many-to-one mapping is employed.

V. Gravatar emails re-identification

In this section, we first present Gravatar, an avatar that can be recognized globally over different web services and in the sequel, we present a re-identification attack due to Bongard [10] to re-identify the email address of a user.

A. Description

Gravatar[1], a portmanteau of *globally recognized avatar*, is a service that allows a member of forums and blogs to automatically have the same profile picture on all participating sites (where the member is registered with the same email address).

Several prominent web services such as GitHub, Stack Overflow and WordPress among others employ Gravatar. The service is available in the form of a plugin and as an API [7], and hence can be easily incorporated into any web service. Moreover, Gravatar support is provided natively in WordPress as of version 2.5 and in web-based project management application Redmine[2] with version 0.8 and later. Support for Gravatars is also provided via a third-party module in the Drupal[3] web content management system.

Gravatars work in the following way: a user first creates an account on gravatar.com using his/her email address, and uploads an avatar to be associated with the account. He/She may optionally enter a variety of profile information to associate with the Gravatar account. This information together with the avatar is then openly-accessible by any participating web service.

For instance, whenever the user interacts with a participating service that requires an email address, such as writing a comment on Stack Overflow, the service checks whether that email address has an associated avatar on gravatar.com. If so, the Gravatar is shown along with the comment.

Gravatar requests: In order to retrieve the Gravatar of a user, the web service makes a request to gravatar.com. This requires no authentication, and is based around simple HTTP GET requests. The web service first generates the MD5 digest of the user's email address and then requests for the avatar using the URL: http://www.gravatar.com/avatar/digest, where

[1] https://en.gravatar.com/
[2] http://www.redmine.org/
[3] https://www.drupal.org/

'digest' is the MD5 hash of the email address (16 bytes). See Figure 4 for a schematic representation.

In case the Gravatar is required to be served over SSL, the URL for the request is modified to: https://secure.gravatar.com/avatar/digest. Gravatar also provides a number of built-in options which allow a developer to configure different parameters such as pixel, default image, *etc*. The profile of a user can also be accessed using a similar process to requesting images. The request URL in this case is: http://www.gravatar.com/digest.

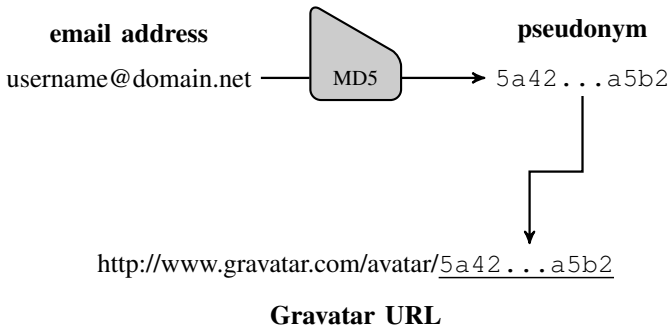

Fig. 4: Gravatar URL generation from an email address.

B. Re-identification

Gravatar often allows users to register and interact with a web service as a pseudonymous account based on their email address. However, the Gravatar request URL containing the MD5 digest is included in the web page and hence is public. A privacy attack on Gravatars consists in re-identifying the anonymous user's email address from the Gravatar digest, *i.e.*, the MD5 digest of the email address. Since, in most of the cases, the email address contains information about the user's name or initials, a user can be de-anonymized by cracking the MD5 digest.

It is worth noticing that Gravatar developers have in the past adverted that hashing (in particular MD5) should be sufficient to obfuscate email addresses. They further argue that emails are harder to crack than passwords since they are longer and less globally different from one another (see [10]). On the contrary, email addresses could be easier to crack than properly generated password. In the following, we discuss why a re-identification attack against Gravatars may succeed.

The first step of the re-identification attack consists in determining if we have a one-to-one or a many-to-one mapping between the email addresses and the pseudonyms.

As specified in RFC 6531 [38], email addresses follow a specific structure: a local part and a domain part separated by the delimiter '@'. The domain part follows the naming rules of Internet domain names while the local part follows the rules specified in RFC 6531. In fact, the local part has a maximum length of 64 characters which are limited to a subset of the ASCII characters: uppercase and lowercase Latin letters (a-z, A-Z); digits (0-9); special characters (# - _ ~ ! $ & ' ( ) * + , ; = :) as well as the period '.' as long as it is not the first or the last character. Other special characters are allowed but only used in specific cases that can be ignored in this study. As for the domain part, RFC 1035 [39] allows for a string of up to 253 ASCII characters.

We first compute the total number of email addresses m. The theoretical value of m can be computed as $m = m_{local} \times m_{domain}$, where m_{local} and m_{domain} are respectively the number of possible local and domain parts. Taking a conservative approach, we consider only local parts composed of lower case Latin letters, digits, and '.', '-', '_' which lead to a total of $m_{local} = (38)^2(39)^{62} \simeq 2^{338}$ distinct values. As for the domain part, we again take a conservative approach by restricting the set of characters to lower case Latin letters, digits, and '-', '_'. The corresponding number of domains is $m_{domain} = 38^{253} \simeq 2^{1328}$.

Hence, under these restrictions, the total number of email addresses is therefore $m = 2^{338} \times 2^{1328} = 2^{1666}$. Also, since the length of an MD5 digest is 128 bits, we have $n = 2^{128}$. As the theoretical value of m satisfies $m \gg n$, the mapping appears to be many-to-one. This implies that the average anonymity set of a Gravatar is $\frac{m}{n} = \frac{2^{1666}}{2^{128}} = 2^{1538}$.

Since a larger anonymity set size generally yields a better privacy, one may be tempted to conclude that the large anonymity set size in case of Gravatar digests should provide very strong anonymity guarantees. While this is indeed true in theory, in practice though, the anonymity set is much smaller.

To see this, let us first note that in the computations detailed above, $m = 2^{1666}$ is the total number of possible email addresses. However, the total number of existing email addresses actually in use is much smaller. To this end, we make the following observations. First, the domain part is in effect restricted to a much smaller space. Indeed, as of today, there are roughly 326 million domain names registered in the world as reported by VERISIGN in its 2016 report [40]. Moreover, according to another recent report [41], the total number of worldwide email accounts is roughly 4.7 billion, which means that the value of m in practice is roughly 2^{32} instead of the previously computed value of 2^{1666}. As a consequence, $m \ll 2^{64}$, and we have a one-to-one mapping between the emails addresses and their pseudonyms.

Second, the domain part of popular email addresses generally fall into a much smaller space, corresponding to big email service providers such as Google (gmail.com), Yahoo! (yahoo.com) or ISPs such as AOL (aol.com), Verizon (verizon.net), Orange (orange.fr). Moreover, some domains are tightly linked to specific regions of the world (*e.g.*, yahoo.co.uk is mainly used in the UK).

Third, the local part of the email address is generally chosen by its owner to be meaningful to others or to clearly identify an individual. As a consequence, they often contain real names, pseudonyms and dictionary words. A common pattern is the concatenation of the firstname and the lastname possibly separated by one of the following delimiters: '.', '-', '_'. For these reasons, despite the theoretically large size of the valid email address space, existing email addresses fall into a much smaller space and are in general predictable and vulnerable to a variety of dictionary-based attacks.

A re-identification attack in practice against Gravatar was demonstrated by Bongard [10]. The goal of the attack was to

identify the email addresses of anonymous commentators on a French political blog (fdesouche.com). To this end, Bongard developed a custom crawler to acquire around 2,400 MD5 hashes of the commentators. The author then generated a dictionary of major email providers (including those providing disposable email addresses), cracking dictionaries (French and English names, Wikipedia entries, sports teams, places, numbers, birth years and postal codes), cracking rules (*e.g.*, most email addresses follow patterns such as firstname.lastname, firstname_lastname, *etc.*) and a cracking rig.

The dictionary coupled with the password-cracking tool `oclHashcat` [42] allowed to recover 45% of the email addresses. The author using another custom email cracking software based on a Bloom filter [43] was able to recover 70% of the email addresses (an increase by 25%). The disadvantage of employing Bloom filters is that the cracking software now generates false positives, *i.e.*, it may claim to have cracked an MD5 digest while in effect it has not.

The idea of recovering email addresses from Gravatar digests was previously demonstrated in 2008 by a user called Abell[4] on developer.it — it was possible to recover 10% of the addresses of 80,000 Stack Overflow users.

VI. MAC Address Anonymization

In this section we consider the case of anonymization of MAC address using hash functions, which is a method adopted by the Wi-Fi tracking industry as a measure to protect privacy [8]. We show how this pseudonymization method can be trivially reversed using simple tools.

A. MAC Address

A MAC address is a 48-bit identifier uniquely allocated to a network interface and by extension to a networked device. Since it uniquely identifies a device and its user, it is a personally identifiable information and is therefore considered as sensitive by the European commission [44]. As they are persistent and globally unique, MAC addresses are collected for tracking purposes by mobile applications [45] and Wi-Fi tracking systems [46], [47].

B. Wi-Fi Tracking Systems

Wi-Fi tracking systems monitor human activities by passively collecting information emitted by portable devices in the physical world. More specifically those systems rely on the MAC addresses of Wi-Fi devices. Most portable devices having their Wi-Fi interface enabled periodically broadcast packets containing their MAC address in order to discover nearby Wi-Fi access points. As a consequence, Wi-Fi tracking systems store mobility traces of individuals along with a unique identifier, the MAC address, that could be potentially linked back to the owner of the device. Applications of Wi-Fi tracking include physical analytics for brick and mortar retail [47] and road monitoring [46].

[4]http://bit.ly/2aMN007

Wi-Fi tracking systems have been identified as a potential privacy threat [15], [16], [48], as they passively collect presence and mobility data on subjects without their consent. In an attempt to protect the privacy of individuals, privacy enhancing mechanisms have been proposed [13]. One of the most popular solutions is the application of a cryptographic hash function to anonymize the MAC address, *i.e.*, instead of storing a MAC address, the system stores its digest. This solution has the advantage of preserving a unique identifier for each individual (given the extremely low collision probability of those hash functions), while supposedly being impossible to be *"reverse-engineered [...] to reveal a device's MAC address."* [8].

C. One-to-One Anonymization

The first attempt to anonymize MAC addresses was made using the SHA-1 hash function [8]. Since, a MAC address is 48 bits long, we have $m = 2^{48}$ and $\ell = 160$ (length of a SHA-1 digest). For these parameters, we have a unique identifier associated to a given pseudonym. Moreover, since m is enumerable in reasonable time, an adversary can easily mount the re-identification attack.

We have reconstructed the re-identification attack using `oclHashcat` [42] and two GPU setups: one with an integrated GPU and the other with a dedicated ATI R9 280X GPU. Table II displays the result of our experiments along with results from the benchmark of `oclHashcat`. Using the ATI R9 280X GPU, 2.6 days were enough to compute the hashes of all the possible MAC addresses. This shows that an exhaustive search is practical with off-the-shelf hardware and freely available software.

TABLE II: Computation time for 2^{48} SHA-1 digests using our own hardware (the first two rows) and the benchmark results from `oclHashcat` (the remaining rows). The number of hashes per second provides the average performance of the cracking tool on the concerned hardware. Cells marked * are estimated results. We used the number of hashes per second to estimate the time required to hash 2^{48} MAC addresses.

Hardware	**#** 10^6 **hashes/s**	**Time** (days)
Integrated GPU	11	296*
ATI R9 280X	1228	2.6
NVIDIA Quadro 600	80	41*
NVIDIA GTX 560 Ti	433	7.5*
NVIDIA GTX 570	629	5*
AMD HD 7970	2136	1.5*
AMD HD 6990	3081	1*

It is even possible to speed up the attack by taking into account the structure of MAC addresses (Figure 5). In order to guarantee the global uniqueness, MAC addresses are allocated to vendors in a range of 2^{24}, which is identified by a OUI (Organizationally Unique Identifier) prefix corresponding to the first three bytes of a MAC address. The remaining 3 bytes correspond to the network interface controller (NIC) which identifies an interface within a given OUI range.

Currently, 22,317 OUI prefixes have been allocated by IEEE (the list is publicly available [49]). It means that only 0.1% of all the possible MAC addresses can be encountered in practice.

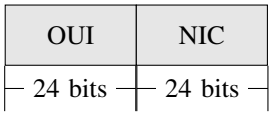

Fig. 5: Structure of a MAC address.

This reduces the exhaustive search from 2^{48} to $22317 \times 2^{24} \approx 2^{38}$ hash computations.

The discrimination attack for a single MAC address is straightforward. It simply consists in computing the digest of the MAC address and checking if it belongs to a specified set of interest. Moreover, one can also exploit the structure of MAC addresses to discriminate devices. To this end, let us consider a scenario in which we have a database of MAC addresses corresponding to Wi-Fi devices in proximity to a certain location. The database is assumed to have been anonymized using a hash function. We further assume that the adversary knows the location from where the pseudonyms have been collected (such as a train station).

In order to re-identify the MAC addresses for a set of pseudonyms, the adversary exploits the distribution of OUIs among different vendors. Figure 6 shows the vendors having the highest number of OUIs registered to IEEE. Relying on the results of Figure 6, she can safely assume that most of these pseudonyms correspond to that of smartphones and Wi-Fi routers. Therefore, instead of mounting a brute force search with all existing OUIs, she first tries the MAC addresses with a OUI associated to popular vendors: Cisco, Apple and Samsung. Moreover, if the adversary wishes to know if a pseudonym corresponds to a Cisco device, then she needs to test 618×2^{24} MAC addresses. In case, she wishes to know if the pseudonym corresponds to a popular smartphone vendor, in particular Apple or Samsung, she would need to test $(474 + 317) \times 2^{24} = 791 \times 2^{24}$ MAC addresses.

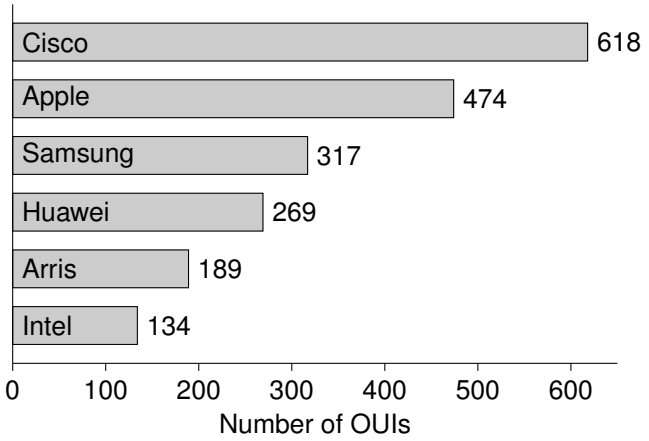

Fig. 6: Number of OUIs per vendor (top 6).

In Table III, we present the time required to learn whether a pseudonym corresponds to a given vendor. The results show that the top six vendors can be discriminated within 8 seconds. Clearly, for less popular vendors, the number of acquired OUIs should be much lesser than that of the top six and hence the required time to discriminate them should be even lower.

TABLE III: Time to discriminate the top 6 vendors in the OUI list using ATI R9 280X GPU.

Vendor	**Time** (s)
Cisco	8
Apple	7
Samsung	5
Huawei	4
Arris	3
Intel	2

The attack reasoning is not limited to smartphones and network equipments. It can be extended to any device from the Internet-of-Things starting with drones (Parrot[5] has five OUIs and DJI[6] has one) and Wi-Fi based fire alarm system (1 OUI for Firepower system[7]).

We note that the de-anonymization attack against MAC addresses (using `oclHashcat` tool) is similar to password cracking. In the case of password cracking, an adversary has access to a database of password digests and her goal is to recover the passwords from the digests. Although our re-identifiation attack and the password cracking mechanism are similar, the original motivation for using a hash function is different: anonymization in one case, and secure credential storage in the other.

D. Many-to-One Anonymization

We now consider the case of anonymizing MAC addresses with truncated digests such that $m \gg 2^{\ell}$. We work with $m = 2^{38}$ (a consequence of the total number of allocated OUIs) and we assume that $\ell = 20$ (SHA-1 digests truncated to 20 bits). The parameters lead to a setting in which $m > n\log(n)$, hence a good approximation for the lower bound of the anonymity set size is $m/2^{\ell} = 2^{18}$. Intuitively, having a large lower bound of 2^{18} should guarantee an acceptable anonymity. However, we explain in the following why this intuition is wrong.

Let us consider an example anonymity set associated to the pseudonym `0xfffff`. It contains the addresses `000668CF92DF` and `000E5B53A051` among others. This is obtained by employing a brute force search on 2^{38} MAC addresses. The first address corresponds to a product of Vicon Industries Inc[8]. The vendor sells IP cameras for surveillance. A manual check on Vicon's website reveals that all their cameras are powered over the Ethernet. The second address is a product of ParkerVision[9] that provides radio frequency solutions and wireless devices.

Now, for the anonymity set to be viable, one must have the pseudonym `0xfffff` to be associated to `000668CF92DF` and `000E5B53A051` with the same probability. However, it is clear that in a database corresponding to wireless devices, the address `000668CF92DF` has a probability 0 to occur. The opposite is true if we consider a database of wired devices. This auxiliary information can be explicitly given with the database or can be acquired by an adversary.

[5] http://www.parrot.com
[6] http://www.dji.com/
[7] www.firepower.ca
[8] http://www.vicon-security.com/
[9] http://parkervision.com/

To extend our example, we need to add additional semantics to MAC addresses. We classify the OUI prefixes depending on the type of products the vendor manufactures. We define four labels: wireless, wired, both and unknown depending on whether the vendor manufactures wireless and/or wired devices. A label can be given to a vendor upon exploring the webpage of the vendor when available. Otherwise, it can be based on the first result returned by a search engine. We employ Microsoft Bing[10] in our study as the search result can be obtained in the RSS format which makes the result easy to parse. To the best of our understanding, GOOGLE search engine does not provide this feature and hence the search response becomes difficult to parse.

Currently, 22,317 OUI addresses are associated to 17,365 vendors. Moreover, we have 92 OUI addresses associated with an unknown vendor name (private in the database). We crawled the websites of these vendors whenever possible and built a database of their labels. We do so by crawling a maximum of 6 pages (chosen arbitrarily to not overload the server) from the fully qualified domain name corresponding to the vendor or to that of the first result returned by Bing. Our database has labels for 12,201 vendors in total (70% of 17,365). The breakdown for the labels is shown in Figure 7.

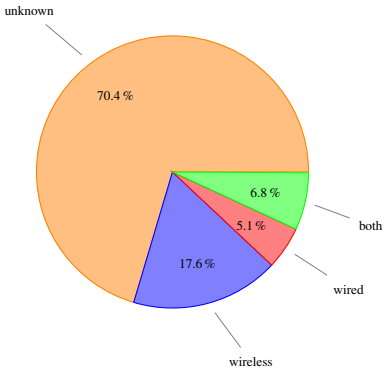

Fig. 7: Percentage of vendors of each type.

Continuing with our previous example, the anonymity set of the pseudonym `0xfffff` contains 14,103 MAC addresses which are associated to wired devices. Theses addresses are unlikely to be associated with the pseudonym `0xfffff` in a wireless related database. Hence, the anonymity set size is reduced from 2^{18} to $2^{18} - 14103 = 24801 = 2^{17}$, *i.e.*, a size reduction by a factor of 2. It is certainly possible to reduce it further by improving the classification and by adding more semantics to OUI prefixes, for instance the country in which the vendor is based.

These results show that controlling the anonymity set size is very difficult. We certainly can not assume $m = 2^{38}$, rather it must be adapted to the dataset we need to anonymize. Additionally, data custodians must also choose the correct ℓ that yields an acceptable anonymity set size for a given dataset.

[10]http://www.bing.com/

VII. GOOGLE SAFE BROWSING

GOOGLE Safe Browsing (GSB) [9] is a browser feature that scans URLs visited by a user for signs of malware and phishing. The service maintains a blacklist of known malware and phishing sites, and whenever a user attempts to visit a web page, the corresponding URL is checked for its belonging in the list. GSB can then warn and dissuade end users from visiting a potentially malicious URL. The browser does so by displaying an interstitial warning page before the suspicious web page is actually requested (see Figure 8).

YANDEX Safe Browsing (YSB) [50] is another GSB like service developed by YANDEX and employs the same GSB architecture. Due to the underlying similarity, we do not distinguish the two services and hence they will be subsumed under the general name of SB services.

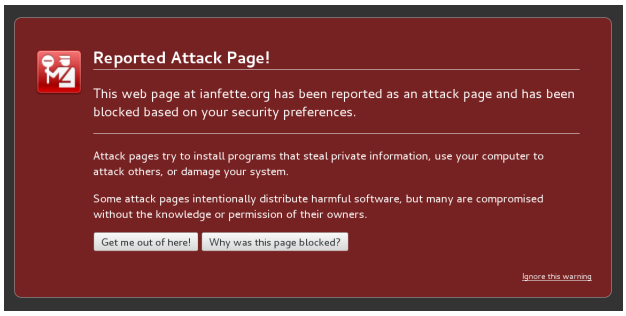

Fig. 8: Warning page displayed to the user for a web page hosting malware.

GOOGLE provides two APIs to query the blacklist. The simplest API is called the `Lookup API v4`. Using this API, a client (typically a browser) can send URLs to check using an `HTTP GET` or `POST` request and the server performs a lookup in the blacklist. This clearly violates the privacy of each user and hence GOOGLE also provides a more complex API called the `Update API v4`. This API was released in June 2016, and all the previous versions of the API have been declared deprecated.

A. *Safe Browsing Update API (v4)*

The `Update API` has been designed to be privacy friendly and hence the underlying architecture is more involved than the `Lookup API`. First, each URL in the blacklist on the server side is hashed using SHA-256 [25] and then the digest is truncated to its first 32 bits, also known as a *prefix*. The resulting list of prefixes is stored on the client's side.

This local database acts as a cache to reduce the number of queries made to the SB server. We note that the blacklist on the server side is extremely dynamic. This is due to the fluctuating behavior of malicious domains: a safe-to-navigate domain transforming into a malicious one and vice versa. A safe-to-navigate domain turns malicious when attackers inject malicious code into it, while a malicious domain becomes safe when harmful codes get cleaned up. Consequently, this requires the client to regularly update the local copy accordingly and hence the name `Update API`.

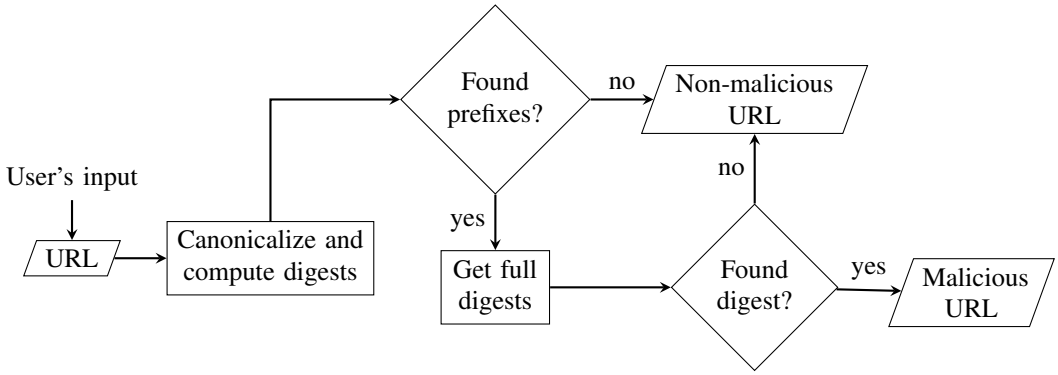

Fig. 9: GOOGLE Safe Browsing `Update API`: Client's behavior flow chart.

Querying for a URL in the `Update API` is very different from the `Lookup API`. The client now does not handle URLs directly. Instead, URLs are first canonicalized following the URI specifications [51]. Canonicalization essentially removes certain parts of a URL such as the username, the password, the port number *etc.*, and sanitizes the URL by removing certain escape characters.

The next step is to generate all the *decompositions* of the URL. A decomposition is a URL composed of sub-domains and sub-paths of the target URL. For the sake of illustration, let us consider the most generic canonicalized HTTP URL of the form `http://a.b.c/1/2.ext?param=1` (see [51], [52]), where, `a.b.c` is a fully-qualified domain name, `1/2.ext` is the URL path and `?param=1` is the query. All the possible decompositions of the URL in the order they are generated are given below:

```
1  a.b.c/1/2.ext?param=1
2  a.b.c/1/2.ext
3  a.b.c/1/
4  a.b.c/
5  b.c/1/2.ext?param=1
6  b.c/1/2.ext
7  b.c/1/
8  b.c/
```

For each decomposition, the client next computes a SHA-256 digest. The digest is then checked for a matching prefix against the locally stored database which contains 32-bit prefixes of malicious URL digests. If no matching prefix is found for any of the decompositions, then the URL can be considered safe. However, if there is a match, the queried URL may not necessarily be malicious: it can be a false positive. A false positive may occur because of collisions on the prefix. In order to eliminate the ambiguity, the client queries the SB server by sending the prefix. The server in response sends all the full digests corresponding to the received prefix. Finally, if the full digest of the client's prefix is not present in the list returned by the server, the URL can be considered safe. Figure 9 summarizes a request through the `Update API`.

We note that the client performs a lookup for decompositions in the given order. The lookup for all the decompositions is required since the complete URL might not have been included in the blacklists. If any of the decompositions is present in the local database, then the initial link is considered as suspicious and the prefix can be forwarded to the SB server for a confirmation. If there are more than 1 matching prefixes, then all the corresponding prefixes are sent.

B. Analysis of Many-to-One Anonymization

It is apparent that the `Update API` is more privacy friendly than the `Lookup API` as clients exchange data with the server using hashed URLs instead of URLs in clear. In fact, the anonymization technique employed in the `Update API` can be seen as a combination of hashing and truncation. Hashing in the form of SHA-256 is used to create pseudonyms for URLs.

Generating pseudonyms (digests) of the URLs however does not suffice to anonymize the data. This is because re-identification and discrimination attacks should be feasible as any motivated adversary can crawl URLs on the web within a reasonable time with the help of a web crawler. In order to prevent these attacks, truncation is applied on the pseudonyms to generate prefixes and force collisions. Truncation of pseudonyms ensures that several URLs share the same reduced pseudonym (prefix).

The previous argument motivates the applicability of the anonymity set size model to measure the achieved privacy. In fact, one may argue that there is an infinite number of pre-images for a 32-bit prefix, hence the anonymity set size should be infinitely large. However, the crucial point here is that the total number of URLs on the web is finite and hence the anonymity metric can at most be finitely small.

Indeed, the most recent statistics published by GOOGLE in 2013 suggest that there are 60 trillion unique URLs [53]. This implies that the average anonymity set size is 14,757. Moreover, since the web is ever growing, the anonymity set size also grows linearly with its size. Hence, in the average case, it is hard for GOOGLE to re-identify a URL from a single 32-bit prefix.

We now study two interesting scenarios typical to SB which may help reduce this anonymity set size. The first concerns the fact that that GOOGLE may include several prefixes (in the local database) for a domain. Second, and even worse, GOOGLE often includes multiple prefixes for a URL. These prefixes may correspond to the different decompositions of the URL. In the following two sections, we discuss these two scenarios in further detail and in the sequel we study their impact on privacy.

1) Multiple prefixes for a domain: Let us first understand why the local database may contain several prefixes for a domain. In fact, it becomes necessary when a domain has

a subset of sub-domains and URL paths which host several malicious URLs. Then, the sub-domains and the paths can be blacklisted instead of including each malicious URL in the database. This approach saves memory footprint on the client's side. We note that the SB service could possibly include only the prefix of the domain to blacklist all its malicious sub-domains and paths. However, this approach also blacklists all non-malicious URLs on the domain. Whence, multiple prefixes are indispensable to prevent certain URLs from being flagged as malicious.

Figure 10 presents a simple page hierarchy for a domain `b.c`, where the colored nodes (for `d.b.c`, `a.b.c/1` and `a.b.c/2`) represent the malicious resources. A naive way to flag the colored nodes as malicious would be to include the corresponding prefixes in the local database. However, as `a.b.c` has only two web pages and both being malicious, therefore, `a.b.c` can itself be flagged as malicious instead of flagging the two web pages that it hosts. Also, `d.b.c` must be flagged as malicious. Hence, in total only two prefixes are required instead of three. We note that if the entire domain, `b.c` is blacklisted instead, then it will also blacklist the benign sub-domain `e.b.c`.

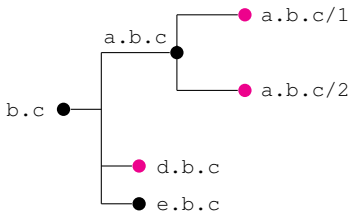

Fig. 10: A sample domain hierarchy for `b.c`. Colored nodes represent malicious resources. A real world example can be google.com for `b.c`; mail.google.com for `a.b.c`; analytics.google.com for `d.b.c`; and maps.google.com for `e.b.c`.

2) Multiple prefixes for a URL: The inclusion of multiple prefixes for a URL is not a hypothetical situation. Instead, experiments with the databases show that GOOGLE and YANDEX indeed include multiple prefixes for a URL. In order to know how many URLs generate more than 2 matches in the blacklists, we first recover the prefix lists of GOOGLE and YANDEX using their respective APIs.

We employ the Alexa list [54] and BigBlackList [55] as test vectors for the experiments. The Alexa list contains a list of 1 million most popular domains. While, BigBlackList contains 2,645,858 URLs. These URLs correspond to phishing, malware and other categories such as pornography, crime, *etc*. The Alexa list has been used in the experiments to determine if GOOGLE or YANDEX indulge in any abusive use of SB.

In case of BigBlackList, we found 103 URLs creating 2 matches in the YANDEX prefix lists. Moreover, we found one URL which creates 3 matches and another one which creates 4 matches. The results on the Alexa list are particularly interesting. We found 26 URLs on 2 domains that create 2 matches each in the malware list of GOOGLE. As for the phishing list, we found 1 URL that creates 2 matches. For YANDEX, we found 1,352 such URLs distributed over 26 domains. 1,158 of these URLs create matches in the YANDEX malware list while the remaining 194 are matches in the YANDEX pornography list. We present a subset of these URLs in Table IV. These URLs are spread over several domains which shows that YANDEX actively includes several prefixes for a URL. This is however less evident for GOOGLE.

It is important to see that (cf. Table IV) including only the prefix for `xhamster.com` suffices to blacklist both `fr.xhamster.com/` and `nl.xhamster.com/`. Adding additional prefixes for the French or the Dutch webpage is completely redundant. Even worse, we later show including more than one prefix for a URL may in fact allow malicious SB servers to mount re-identification attacks.

Whenever, prefixes corresponding to multiple decompositions for a URL/domain are included in the local database, the URL/domain becomes less anonymous. The key idea is the following: Randomly choosing two URLs for which the decomposition matches exactly the same 2 prefixes is unlikely, with probability $\frac{1}{2^{64}}$. It implies that if there is more than one match for a URL, then it can be re-identified using its prefixes.

In the sequel, we develop further this intuition and analyze whether multiple prefixes may indeed allow GSB and YSB to re-identify the URL visited by a client. In order to present a comprehensible privacy analysis, we henceforth consider the simplified case of 2 prefixes. The analysis for the case when the server receives more than 2 prefixes per URL follows in a straightforward manner.

3) Collisions on two prefixes: As in the single prefix case, in theory more than two distinct URLs may yield the same two prefixes. The larger is the number of such URLs, the more difficult is the re-identification. These URLs exist due to three possible types of collisions on the prefixes.

In order to illustrate the different possible collisions, we present a set of examples in Table V. We assume that the client visits the target URL `a.b.c` and hence the server receives the corresponding two prefixes, denoted by A and B (we assume that they are included in the local database). The server using these prefixes must determine the exact URL visited by the client.

The next 3 URLs exemplify the different collisions. In the first type (Type I), several distinct yet "related" URLs share common decompositions and these decompositions yield the shared prefixes. We refer to two distinct URLs as "related" if they have common sub-domains.

The second type of collisions (Type II) is due to distinct yet "related" URLs that share one decomposition and hence one common prefix, while the other common prefix is due to the collision on truncated digests. Finally, the last type of collisions (Type III) appears when completely "unrelated" URLs generate the same prefixes. The latter may occur again due to collisions on the truncated digests. In the following, by a Type I URL, we mean a URL that generates a Type I collision with a given URL. We similarly define Type II and Type III URLs for a given URL.

Clearly, $\mathbb{P}[\text{Type I}] > \mathbb{P}[\text{Type II}] > \mathbb{P}[\text{Type III}]$, where $\mathbb{P}[X]$ denotes the probability of an event X. Under the uniformity assumption of hash functions, a Type III collision is highly unlikely, with a probability of $\frac{1}{2^{64}}$.

TABLE IV: A subset of URLs from the Alexa list creating multiple matches in the GOOGLE and YANDEX database.

	URL	matching decomposition	prefix
GOOGLE	http://wps3b.17buddies.net/wp/cs_sub_7-2.pwf	17buddies.net/wp/cs_sub_7-2.pwf	0x18366658
		17buddies.net/wp/	0x77c1098b
	http://www.1001cartes.org/tag/emergency-issues	1001cartes.org/tag/emergency-issues	0xab5140c7
		1001cartes.org/tag/	0xc73e0d7b
YANDEX	http://fr.xhamster.com/user/video	fr.xhamster.com/	0xe4fdd86c
		xhamster.com/	0x3074e021
	http://nl.xhamster.com/user/video	nl.xhamster.com/	0xa95055ff
		xhamster.com/	0x3074e021
	http://m.wickedpictures.com/user/login	m.wickedpictures.com/	0x7ee8c0cc
		wickedpictures.com/	0xa7962038
	http://m.mofos.com/user/login	m.mofos.com/	0x6e961650
		mofos.com/	0x00354501
	http://mobile.teenslovehugecocks.com/user/join	mobile.teenslovehugecocks.com/	0x585667a5
		teenslovehugecocks.com/	0x92824b5c

TABLE V: An example target URL a.b.c with different possible collisions. URLs g.a.b.c and a.b.c are "related" since they share two common decompositions, namely a.b.c/ and b.c./.

		URL	Decompositions	Prefixes
	Target URL	a.b.c	a.b.c/ b.c/	*A* *B*
Coll. Type	Type I	g.a.b.c	g.a.b.c/ a.b.c/ b.c/	*C* *A* *B*
	Type II	g.b.c	g.b.c/ b.c/	*A* *B*
	Type III	d.e.f	d.e.f/ e.f/	*A* *B*

We note that for Type I and Type II collisions to occur, the URLs must share at least one common decomposition. The probability of these collisions hence depends on the number of decompositions of URLs hosted on the domain. In general, the smaller is the number of decompositions per URL, the lower is the probability that Type I and Type II URLs exist. Moreover, a Type II URL exists only if the number of decompositions on a domain is larger than 2^{32}. This implies that for small sized domains, Type II URLs should not exist.

As a result, the ambiguity in the re-identification can only arise due to Type I collisions. In the following, we discuss the problem of URL re-identification with a focus on URLs that admit Type I collisions.

4) Re-identification: We note that a target URL with few decompositions has a very low probability to yield Type I collisions, and hence it can be easily re-identified. In case of URLs with a large number of decompositions, the server would require more than two prefixes per URL to remove the ambiguity. Nevertheless, the SB provider can still determine the common sub-domain visited by the client using only two prefixes. This information may often suffice to identify suspicious behavior when the domain in question pertains to specific traits such as pedophilia or terrorism.

Now, let us further analyze the problem of re-identifying URLs for which Type I collisions occur. To this end, we consider an illustrative example of a domain b.c that hosts a URL a.b.c/1 and its decompositions (see Table VI). We assume that these are the only URLs on the domain. The URL generates four decompositions. Two of these decompositions include the domain name 'a' as a sub-domain while the remaining two do not. These decompositions yield prefixes denoted by *A*, *B*, *C* and *D* respectively. We note that the most general canonicalized URL is of the form http://a.b.c/1/2.ext?param=1 of which our example URL is only a slightly simplified form with the query part removed.

TABLE VI: A sample URL on b.c with its 4 decompositions.

URL	Decompositions	Prefix
a.b.c/1	a.b.c/1	*A*
	a.b.c/	*B*
	b.c/1	*C*
	b.c/	*D*

We analyze the following three cases depending on the prefixes sent by the client to the SB server:

- **Case 1.** (A, B): If the server receives these prefixes, it can be sure that the client has visited the URL that corresponds to the first prefix *A*, *i.e.*, a.b.c/1. This is because a.b.c/1 is the only URL that generates two decompositions yielding prefixes *A* and *B*. For instance, the decompositions of the URL a.b.c/ can only generate prefixes *B*, *C* or *D* but not *A*.
  The probability that re-identification fails in this case is $\mathbb{P}[\text{Type III}] = \frac{1}{2^{64}}$. This holds because we assume that the domain b.c hosts only 4 URLs, hence the probability that the re-identification fails is the same as the probability of finding a Type III URL for prefixes (A, B). Our assumption on the number of URLs on the domain b.c ensures that no Type II URLs exist.
- **Case 2.** (C, D): In this case, the possible URLs that the client could have visited are: a.b.c/1, a.b.c/ or b.c/1. This is because these are the only URLs which upon decomposition yield both *C* and *D*. These URLs correspond to prefixes *A*, *B* and *C* respectively. Hence, in order to remove the ambiguity and re-identify the exact URL visited by the client, the SB provider would include additional prefixes in the local database. If it includes the prefix *A*, in addition to *C* and *D*, then it can learn whether the client visited the URL a.b.c/1 or b.c/1. More precisely, if the client visits a.b.c/1 then prefixes *A*, *C* and *D* will be sent to the server, while if the client visits b.c/1, then only *C* and *D* will be sent. Similarly, to distinguish whether the client visits a.b.c/ or b.c/,

the SB provider would additionally need to include B.

- **Case 3.** One of $\{A,B\}\times\{C,D\}$: If the prefix A creates a match, then the visited URL is `a.b.c/1`. This is because `a.b.c/1` is the only URL that upon decomposition can yield the prefix A. If the prefix B creates a match, then the client has either visited `a.b.c/1` or `a.b.c/`. This is again because these are the only two URLs capable of generating the prefix B. As in Case 2, to distinguish between these two URLs, the SB provider is required to include an additional prefix A in the prefix database.

As a general rule, decompositions that appear before the first prefix are possible candidates for re-identification. Hence, lower-level domain names and URL paths can be re-identified with a higher certainty than the ones at a higher level.

To this end, we consider the case of *leaf* URLs on a domain. We call a URL on a given domain a *leaf*, if it does not belong to the set of decompositions of any other URL hosted on the domain. A leaf URL can also be identified as a leaf node in the domain hierarchy (see Figure 11). Type I collisions for these URLs can be easily eliminated during re-identification with the help of only two prefixes. The first prefix corresponds to that of the URL itself, while the other one may arbitrarily correspond to any of its decompositions. In the example of Table VI, the URL `a.b.c/1` is a leaf URL on the domain `b.c`, and hence it can be re-identified using prefixes (A,B).

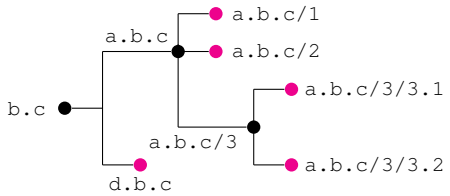

Fig. 11: A sample domain hierarchy for `b.c`. Colored nodes are leaf URLs: `a.b.c/1`, `a.b.c/2`, `a.b.c/3/3.1`, `a.b.c/3/3.2` and `d.b.c`. A real world example can be google.com for `b.c`; mail.google.com for `a.b.c` and analytics.google.com for `d.b.c`.

Clearly, only non-leaf URLs contribute to Type I collisions. Hence, in order to re-identify non-leaf nodes, one must include more than two prefixes per node.

Gerbet *et al.* [22] perfom extensive experiments with two datasets to understand the distribution of decompositions over domains and the rate of Type I and Type II collisions. The first dataset consists of 1 million domains from the Alexa list, while the other contains 1 million random domains. Their experiments could not find any Type II collision. Moreover, they observed that 56% of the domains in the random dataset do not have Type I collisions, while the same is true for around 60% of the domains in the Alexa dataset. Hence, URLs on these domains can be easily re-identified using only two prefixes. For the others, SB servers would require more than two prefixes.

C. One-to-One Anonymization Problem

SB presents a unique case study since apart from employing a many-to-one anonymization, it also incorporates a one-to-one anonymization scheme. This is because the malicious URLs on the server side are stored in the form of SHA-256 digests. Our goal in this section is to study this aspect of SB services. To this end, we attempt to re-identify the URLs which correspond to the full SHA-256 digests in the blacklists provided by GOOGLE and YANDEX.

We harvested phishing and malware URLs, domains, and IP addresses from several sources and tested for their belonging to the blacklists. The list of all our sources can be found in [56]. We also harvested 1,288,503 malware URLs, 38,138 phishing URLs and 2,645,858 URLs of other categories from BigBlackList [55]. Lastly, we obtained 106,928,034 second-level domains (SLDs) from the DNS Census 2013 project [57]. The project provides a public dataset of registered domains and DNS records gathered in the years 2012-2013. We included the last dataset to determine the percentage of prefixes in the local database that correspond to SLDs. A summary of all the datasets employed in our analysis is given in Table VII.

TABLE VII: Dataset used for inverting SHA-256 digests.

Dataset	Description	#entries
Malware list	malware	1,288,503
Phishing list	phishing	38,138
BigBlackList	malw., phish., porno, others	2,645,858
DNS Census-13	second-level domains	106,928,034

Our experimental results are shown in Table VIII. We observe that reconstruction using Malware, Phishing and BigBlackList datasets is largely inconclusive, except for adult-related and pornographic entries using BigBlackList. For the adult-related entries, the reconstruction rate using BigBlackList is 16%, while for the pornographic ones, it is 44%. The DNS Census-13 dataset produces a much better reconstruction for all the lists in general. The rate is as high as 53% for YANDEX files.

Phishing domains are short-lived and since the DNS Census-13 dataset dates back to 2013, the result of the reconstruction for phishing lists is very limited, only 0.1% for GOOGLE and 1.1% for YANDEX. A side consequence of the re-identification attack is that it also reveals potentially vulnerable domains to which future malware attacks can be targeted.

It is pertinent to compare the result of our reconstruction with a similar attempt with another list in the past. German censorship federal agency called BPjM maintains a secret list of about 3,000 URLs believed to be unsuitable for women and children. The list is anonymized and distributed in the form of MD5 or SHA-1 hashes as the "BPJM-Modul" [14]. Though similar to the lists handled by GOOGLE and YANDEX, hackers have been able to retrieve 99% of the cleartext entries.

We have applied the same approach, yet the reconstruction rate obtained has not been equally high. This proves that in order to reconstruct the database in clear, one would need high crawling capabilities and hence it is not achievable for general users. Furthermore, unlike the BPjM list, the blacklists in GSB and YSB are extremely dynamic. This requires a regular crawling of the web, which renders the reconstruction even more difficult.

TABLE VIII: Matches found in each blacklist for entries of our datasets. Cells marked 0* have a percentage very close to zero. For these cells we give the exact number of matches found in parentheses.

		%match			
	list name (# entries)	**Malware list**	**Phishing list**	**BigBlackList**	**DNS Census-13**
GOOGLE	goog-malware-shavar-full (317785)	6.7	0*(26)	0.6	17.4
	googpub-phish-shavar-full (312298)	2.3	0.1	0*(6)	0.1
YANDEX	ydx-malware-shavar-full (279039)	1.2	0.03	2.7	28.7
	ydx-adult-shavar-full (250)	9.2	0	16	45.2
	ydx-mobile-only-malware-shavar-full (1977)	1	0	1	35
	ydx-phish-shavar-full (268)	0	1.1	0	1.1
	ydx-porno-hosts-top-shavar-full (99750)	1.6	0*(8)	44	53.3
	ydx-sms-fraud-shavar (447)	1.8	0	0	7.6

VIII. CONCLUSION

This tutorial explains the limits of cryptographic hash functions to create pseudonyms. However, it does not include three natural directions when considering hashing and privacy: 1) memory-hard cryptographic primitives such as `scrypt` [58] or computationally slow hash functions such as `bcrypt` [59], 2) keyed-hash functions, and 3) privacy-friendly solutions built atop hash-based data structures such as Bloom filters [43] and CM sketches [60].

The first approach to mitigate re-identification and discrimination attacks could be to use a memory-hard or a computationally slow pseudo random function. Some of these functions such as `scrypt` [58] (a password-based key derivation function) and `bcrypt` [59] (a password hashing function) have become very popular recently. They can be useful to mitigate the attacks described in this tutorial. However, they do not completely prevent the attacks and they need the test to time to assess their security.

A more advanced application of hashing for anonymization would be to use a keyed-hash function *aka* a hash-based message authentication code (HMAC). Employing an HMAC may counter the re-identification and discrimination attacks since the underlying key cannot be guessed by the attacker. However, use of a key drastically restricts its deployment as it should be securely stored and might need to be exchanged with a possibly malicious entity. For instance, using a unique key in SB cannot work as an adversary will also have access to it. While, maintaining a key for each user is clearly not scalable. Moreover, since an HMAC is deterministic, two records for the same identifier can be linked. Other attacks exploiting auxiliary information have also been identified in the past [1].

As for other data structures, Bloom filters have particularly become a popular choice in designing privacy-friendly solutions [61]–[63]. Two notable examples are BLIP [63] and RAPPOR [62] which allow privacy preserving data mining on statistical databases. In BLIP, the main objective is to privately compute in a distributed manner the similarity between user profiles by relying only on their Bloom filter representations. While RAPPOR is a tool developed by GOOGLE to gather usage statistics.

The idea underpinning these tools is to sufficiently randomize the data structure to obtain the desired utility/privacy. This eventually leads to a much stronger privacy notion (than the one considered in this tutorial) called *differential privacy* [64]. Differential privacy is undoubtedly the most promising of all state-of-the-art privacy notions for anonymization. However, it suffers from the disadvantage that the eventual utility is application sensitive, which is otherwise not the case with hash functions.

Acknowledgment. This research was partially supported by the Labex PERSYVAL-LAB (ANR-11-LABX-0025-01) funded by the French program Investissement d'avenir.

SELECTED REFERENCES

[1] "Opinion 05/2014 on Anonymisation Techniques," 2014, http://ec.europa.eu/justice/data-protection/index_en.htm.
[2] "Opinion 2/2010 on online behavioral advertising," 2010, http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2010/wp171_en.pdf.
[3] "Opinion 16/2011 on EASA/IAB Best Practice Recommendation on Online Behavioural Advertising," 2011, http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2011/wp188_en.pdf.
[4] "Privacy Technology Focus Group: Final Report and Recommendations," 2006, http://www.it.ojp.gov/documents/privacy_technology_focus_group_full_report.pdf.
[5] "Directive concerning the processing of personal data and the protection of privacy in the electronic communications sector," 2002, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:en:HTML.
[6] D. Goodin, "Poorly anonymized logs reveal NYC cab drivers' detailed whereabouts," ArsTechnica, Jun. 2014, accessed 21-07-16. [Online]. Available: http://arstechnica.com/tech-policy/2014/06/poorly-anonymized-logs-reveal-nyc-cab-drivers-detailed-whereabouts/
[7] T. Preston-Werner, "Gravatar," 2007, https://en.gravatar.com/site/implement/.
[8] "Euclid Analytics - Privacy Statement," http://euclidanalytics.com/privacy/statement/, Euclid Inc., 2015, accessed 15/05/15. [Online]. Available: http://euclidanalytics.com/privacy/statement/
[9] Google Inc., "Safe Browsing API," https://developers.google.com/safe-browsing/, 2016.
[10] D. Bongard, "De-anonymizing Users of French Political Forums," 0xcite LLC, Luxembourg, Tech. Rep., October 2013, http://archive.hack.lu/2013/dbongard_hacklu_2013.pdf.
[11] "A Face Is Exposed for AOL Searcher No. 4417749," 2006, http://www.nytimes.com/2006/08/09/technology/09aol.html?_r=0.
[12] "eyeQ Privacy Policy." [Online]. Available: https://www.eyeqinsights.com/privacy/
[13] Future of Privacy Forum, "Mobile Location Analytics Code of Conduct," Future of Privacy Forum, Tech. Rep., Oct. 2013.
[14] BPjM, "BPJM Modul," Online, 2014, http://bpjmleak.neocities.org/.
[15] Parker Higgins and Lee Tien, "Mobile Tracking Code of Conduct Falls Short of Protecting Consumers," Oct. 2013. [Online]. Available: https://www.eff.org/deeplinks/2013/10/mobile-tracking-code-conduct-falls-short-protecting-consumers
[16] Ashkan Soltani, "Privacy trade-offs in retail tracking | Federal Trade Commission," May 2015. [Online]. Available: https://www.ftc.gov/news-events/blogs/techftc/2015/04/privacy-trade-offs-retail-tracking
[17] J. Mayer, "Questionable Crypto in Retail Analytics," http://webpolicy.org/2014/03/19/questionable-crypto-in-retail-analytics/, March 2014, accessed 15/05/15. [Online]. Available: http://webpolicy.org/2014/03/19/questionable-crypto-in-retail-analytics/

[18] Claire Bouchenard, "JC Decaux's pedestrian tracking system blocked by French data regulator," Oct. 2015. [Online]. Available: http://marketinglaw.osborneclarke.com/advertising-regulation/jc-decauxs-pedestrian-tracking-system-blocked-by-french-data-regulator/

[19] D. Koukis, S. Antonatos, D. Antoniades, E. P. Markatos, and P. Trimintzios, "A generic anonymization framework for network traffic," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 5. IEEE, 2006, pp. 2302–2309. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/4024508/

[20] R. Chandra, R. Mahajan, V. Padmanabhan, and M. Zhang, "CRAWDAD dataset microsoft/osdi2006 (v. 2007-05-23)," Downloaded from http://crawdad.org/microsoft/osdi2006/20070523/pcap, May 2007, traceset: pcap.

[21] L. Demir, M. Cunche, and C. Lauradoux, "Analysing the Privacy Policies of Wi-Fi Trackers," in *Proceedings of the 2014 Workshop on Physical Analytics*, ser. WPA '14. New York, NY, USA: ACM, 2014, pp. 39–44.

[22] T. Gerbet, A. Kumar, and C. Lauradoux, "A Privacy Analysis of Google and Yandex Safe Browsing," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 25-July 1*, 2016.

[23] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1996.

[24] Q. Dang, "Recommendation for Applications Using Approved Hash Algorithms," National Institute of Standards & Technology, Tech. Rep. SP 800-107 Revision 1, August 2012.

[25] NIST, "Secure Hash Standard (SHS)," National Institute of Standards & Technology, Tech. Rep. FIPS PUB 180-4, march 2012.

[26] ——, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," National Institute of Standards & Technology, Tech. Rep. FIPS PUB 202, May 2014, draft.

[27] M. J. Wiener, "The full cost of cryptanalytic attacks," *J. Cryptology*, vol. 17, no. 2, pp. 105–124, 2004.

[28] R. C. Merkle, "One Way Hash Functions and DES," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, 1989, pp. 428–446.

[29] I. Damgård, "A Design Principle for Hash Functions," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, 1989, pp. 416–427.

[30] B. Preneel, "The State of Cryptographic Hash Functions," in *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School*, ser. Lecture Notes in Computer Science, vol. 1561. Aarhus, Denmark: Springer, July 1998, pp. 158–182.

[31] ——, "The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition," in *Topics in Cryptology - CT-RSA 2010*, ser. Lecture Notes in Computer Science, vol. 5985. San Francisco, CA, USA: Springer, March 2010, pp. 1–14.

[32] ——, "Cryptographic Hash Functions: Theory and Practice," in *Information and Communications Security - 12th International Conference, ICICS 2010*, ser. Lecture Notes in Computer Science, vol. 6476. Barcelona, Spain: Springer, December 2010, pp. 1–3.

[33] A. Pfitzmann and M. Köhntopp, "Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology," in *International Workshop on Design Issues in Anonymity and Unobservability*, ser. Lecture Notes in Computer Science, vol. 2009. Berkeley, CA, USA: Springer, July 2000, pp. 1–9.

[34] R. Motwani and P. Raghavan, *Randomized Algorithms*. New York, NY, USA: Cambridge University Press, 1995.

[35] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, January 1968, vol. 1.

[36] G. Ercal-Ozkaya, "Routing in Random Ad-Hoc Networkds: Provably Better than Worst-case," Ph.D. dissertation, University of California at Los Angeles, 2008.

[37] A. Serjantov and G. Danezis, "Towards an Information Theoretic Metric for Anonymity," in *Privacy Enhancing Technologies, Second International Workshop, PET 2002*, ser. Lecture Notes in Computer Science, vol. 2482. San Francisco, CA, USA: Springer, April 2002, pp. 41–53.

[38] J. Yao and W. Mao, "SMTP Extension for Internationalized Email," RFC 6531 (Proposed Standard), Internet Engineering Task Force, Feb. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6531.txt

[39] P. Mockapetris, "Domain names - Implementation and Specification," RFC 1035 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[40] Verisign Inc., http://verisigninc.com/en_US/innovation/dnib/index.xhtml, 2016.

[41] The Radicati Group, Inc., "Email Statistics Report, 2015-2019," 2015, http://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf.

[42] J. Steube. (2016) Hashcat Advanced Password Recovery. http://hashcat.net.

[43] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[44] THE WORKING PARTY ON THE PROTECTION OF INDIVIDUALS WITH REGARD TO THE PROCESSING OF PERSONAL DATA, "Opinion 01/2017 on the Proposed Regulation for the ePrivacy Regulation," 2017, ec.europa.eu/newsroom/document.cfm?doc_id=44103.

[45] J. P. Achara, M. Cunche, V. Roca, and A. Francillon, "Short Paper: WifiLeaks: Underestimated Privacy Implications of the access_wifi_state Android Permission," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*, ACM. New York, NY, USA: ACM, 2014, pp. 231–236.

[46] A. Musa and J. Eriksson, "Tracking Unmodified Smartphones Using Wi-Fi Monitors," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, ACM. New York, NY, USA: ACM, 2012, pp. 281–294.

[47] Brian Fung, "How stores use your phone's WiFi to track your shopping habits," Oct. 2013. [Online]. Available: https://www.washingtonpost.com/news/the-switch/wp/2013/10/19/how-stores-use-your-phones-wifi-to-track-your-shopping-habits/

[48] A. Farshidi, "The New Retail Experience and Its Unaddressed Privacy Concerns: How RFID and Mobile Location Analytics are Collecting Customer Information," *Journal of Law, Technology, & the Internet*, vol. 7, no. 1, p. 15, 2016. [Online]. Available: http://scholarlycommons.law.case.edu/jolti/vol7/iss1/13/

[49] IEEE, IEEE, 2015. [Online]. Available: http://standards-oui.ieee.org/oui.txt

[50] Yandex, "Yandex Safe Browsing," http://api.yandex.com/safebrowsing/, 2015.

[51] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (INTERNET STANDARD), Internet Engineering Task Force, Jan. 2005, updated by RFCs 6874, 7320. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[52] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," Internet Requests for Comments, RFC Editor, RFC 1738, December 1994, https://www.ietf.org/rfc/rfc1738.txt.

[53] Google Inc., http://googleblog.blogspot.fr/2008/07/we-knew-web-was-big.html, 2008.

[54] "Alexa 1M Global Sites," Online, 2015, http://bit.ly/1yhXcgL.

[55] "BigBlackList," Online, 2015, http://urlblacklist.com/.

[56] Zeltser Security Corp, https://zeltser.com/malicious-ip-blocklists/, 2015.

[57] "DNS Census," Online, 2013, https://dnscensus2013.neocities.org/.

[58] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function," RFC 7914 (Informational), Internet Engineering Task Force, Aug. 2016. [Online]. Available: http://www.ietf.org/rfc/rfc7914.txt

[59] N. Provos and D. Mazières, "A Future-Adaptable Password Scheme," in *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, June 6-11, 1999, Monterey, California, USA*, 1999, pp. 81–91. [Online]. Available: http://www.usenix.org/events/usenix99/provos.html

[60] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[61] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber, "On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients," in *Annual Computer Security Applications Conference, ACSAC 2014*. New Orleans, LA, USA: ACM, December 2014, pp. 326–335.

[62] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, 2014.

[63] M. Alaggan, S. Gambs, and A. Kermarrec, "BLIP: Non-interactive Differentially-Private Similarity Computation on Bloom filters," in *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*, 2012, pp. 202–216.

[64] C. Dwork, "Differential Privacy," in *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, ser. Lecture Notes in Computer Science, vol. 4052. Venice, Italy: Springer Verlag, July 2006, pp. 1–12.