# Obfuscation of Sensitive Data in Network Flows

Daniele Riboni*, Antonio Villani**, Domenico Vitali**, Claudio Bettini* and Luigi V. Mancini**
* Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano.
Via Comelico 39, Milan 20135, Italy. E-mail: {daniele.riboni, claudio.bettini}@unimi.it
** Dipartimento di Informatica, Università di Roma "La Sapienza"
Via Salaria 113, Rome 00198, Italy. E-mail: {villani, vitali, lv.mancini}@di.uniroma1.it

***Abstract*—In the last decade, the release of network flows has gained significant popularity among researchers and networking communities. Indeed, network flows are a fundamental tool for modeling the network behavior, identifying security attacks, and validating research results. Unfortunately, due to the sensitive nature of network flows, security and privacy concerns discourage the publication of such datasets. On the one hand, existing techniques proposed to sanitize network flows do not provide any formal guarantees. On the other hand, microdata anonymization techniques are not directly applicable to network flows. In this paper, we propose a novel obfuscation technique for network flows that provides formal guarantees under realistic assumptions about the adversary's knowledge. Our work is supported by extensive experiments with a large set of real network flows collected at an important Italian Tier II Autonomous System, hosting sensitive government and corporate sites. Experimental results show that our obfuscation technique preserves the utility of network flows for network traffic analysis.**

## I. Introduction

Recently, there has been a growing interest in releasing large dataset of *network flows* of Internet traffic. Indeed, such flows are a very valuable resource for researchers; for instance, to model the network behavior, to experiment new protocols, and to study security attacks. However, the release of network flows of Internet traffic poses serious concerns to the privacy and the security of the users of computer networks involved. For example, based on the analysis of network flows about the Web sites visited by a given individual, an adversary may infer sensitive data, such as political preferences, religious belief, health status, and so on. Network flows may also reveal personal communications among specific individuals, such as the existence of email exchanges, and chat sessions among them. Moreover, network flows can be exploited by adversaries to gather useful information in planning network attacks; for instance, for identifying possible bottlenecks in the target network in order to increase the impact of a Denial of Service attack.

As a consequence, various research efforts have been carried on to protect privacy while preserving the practical applicative interest in using the released network flows. Early techniques were based on the substitution of the real IP addresses with pseudo-IDs (for instance, in *Crypto-PAn* [1]). However, it has been shown that this technique is insufficient, since an adversary may reconstruct the real IPs based on the values of other fields of the flows [2], exploiting his knowledge of the characteristics of network hosts (*fingerprinting* attacks), or by injecting peculiar flows in the monitored network (*injection* attacks). For this reason, more sophisticated techniques have been proposed, based on the perturbation of other data in the flows (e.g., [3], [4], [5], [6]). However, the techniques proposed till the time of writing do not provide any formal guarantee of protection, and it has been recently shown that they are prone to different kinds of attacks [7]. On the other hand, as we explain in Section II-B, well-known techniques proposed for microdata anonymization are not directly applicable to network flows.

In this paper, we tackle the challenging research issue of sanitizing network traces while preserving the data utility and providing formal guarantees of confidentiality protection. The main contributions of this work are the following:

- we formally model the problem of network flow obfuscation;
- we propose a novel defense technique, named $(k, j)$-obfuscation, and we formally prove that it guarantees protection of data confidentiality under realistic assumptions;
- we present algorithms to enforce $(k, j)$-obfuscation, and we experimentally evaluate our technique with a very large dataset of real network flows; results show that our obfuscation technique preserves the utility of data.

The dataset used for our experiments was collected from an Italian transit tier II Autonomous System (AS). This network is connected to the three main network infrastructures present in Italy (Commercial, Research and Public Administration networks), and to several international providers.

The rest of the paper is structured as follows. Section II discusses related work. In Section III, we formally model network flow obfuscation and the adversary knowledge. In Section IV, we introduce our defense and formally prove its guarantees of confidentiality protection. In Section V, we present the algorithm to enforce our defense. Section VI illustrates the experimental evaluation. Section VII concludes the paper.

## II. Related work

Early techniques for network flow obfuscation were based on the encryption of source and destination IP addresses. However, those techniques proved to be ineffective, since an adversary might be able to re-identify message source and destination by other values in a network flow, or in a sequence of flows (see, e.g., [3], [8], [9], [10]). King et al. in [2] propose an extensive taxonomy of attacks against network flow sanitization methods; techniques fall into two main categories:

- *Fingerprinting:* re-identification is performed by matching flows fields' values to the characteristics of the target environment (such as knowledge of network topology and its settings, types of OS and services of target hosts, etc). Typical re-identifying values for network flows are: Type of Service (tos), TCP Flags, number of bytes, and number of packet per flow.
- *Injection:* the adversary injects a sequence of flows in the network to be logged, that are easily recognized due to their specific characteristics; e.g., marked with uncommon TCP flags, or following particular patterns.

Additional techniques can be used to exploit the results of the above attacks to decrypt IP addresses of new network flows. In particular, if the IP address encryption is performed with the same key across the whole set of flows (as in most existing defense techniques), and the adversary discovers an IP mapping in one flow, he can decrypt the same IP address in any other flow.

### A. Defenses tailored to network flows

Several effort have been devoted to the implementation of frameworks (e.g., [5]) or configurable tools (e.g., [6]) through which the network administrator can define ad-hoc and per-field obfuscation policies. Roughly speaking, defenses that can be found in the literature (e.g., [3], [4] among many others) take a "reactionary" approach: typically, in those works, a new kind of attack is identified, and a defense technique is proposed for that attack, which is generally based on the permutation/generalization of some fields' values. However, proposed techniques do not provide a general solution. Indeed, as theoretically proved by Brekne and Årnes in [9], and empirically shown by Burkhart et al. in [7], those techniques can be easily defeated by the injection of flows following complex patterns over sufficiently long periods of time.

As a case study, we consider the well-known *Crypto-PAn* [1] technique, which is currently incorporated within several network flow collector tools. Crypto-PAn is a sanitization tool for network flows that encrypts IP addresses in a prefix-preserving manner. A malicious user, which acts inside the monitored network, can inject bogus and easily detectable flows in order to understand how one IP address is mapped to its encrypted value inside the obfuscated flow set. Thanks to the fact that each IP address's octet (8 bit length integer value) is always mapped to the same encrypted value, an adversary can obtain the encrypted version of each one of the 255 possible octets values injecting a small number of bogus flows.

The defense technique we propose adopts cryptographic primitives to hide real IP addresses, and obfuscation of flow fields' values. However, differently from previous works, it provides strong confidentiality protection even when the adversary can reconstruct the mapping between an IP address and its encrypted value, possibly as a result of injection attacks.

### B. Microdata anonymization techniques

Techniques proposed in the database area for microdata anonymization have the advantage of providing formal privacy guarantees, under specific assumptions. Hence, it is natural to investigate the application of these techniques to network flows. At first glance, network flow logs seem very similar in nature to any other recordset stored in a relational database (census data, medical records, etc.). However, as we explain below, those techniques are unfeasible to network flows, due to the peculiar characteristics of these data.

The simplest microdata anonymity principle is $k$-anonymity [11], which consists in making any record indistinguishable in a group of at least $k$ records based on *Quasi Identifier (QI)* values; i.e., values that, joined with external information, may reduce the candidate set of record respondents. Any group of records having the same values for QI attributes is called a *QI-group*. The main criticism found in the literature about the application of this principle to network flow logs (see, e.g., [12]) regards loss of information: indeed, since many fields of network flows may act as QI, data quality would be degraded to an unacceptable extent. The same argument holds for more sophisticated privacy principles that guarantee not only anonymity but also sensitive value diversity, such as $l$-diversity [13] and $t$-closeness [14].

However, we observe that the above mentioned principles are not even applicable to the anonymization of network flows. Indeed, if the private value of each individual does not change in released microdata (this is the case of network flow logs, if IP address encryption is consistent across the whole set of flows), works in [11], [13], [14] are effective only under the assumption that each individual is the respondent of at most one record in the released microdata. Indeed, if the adversary knows that the same individual (in our case, an IP address $I$) is the respondent of one tuple in more than one QI-group, an adversary may be able to derive the confidential information (the encryption of $I$) by simply intersecting the private values of tuples in those QI-groups.

Since the same IP address typically appears in multiple network flows, an appropriate privacy principle to be considered is $m$-invariance [15], which has been proposed to enforce both anonymity and diversity for incremental release of microdata. This principle ensures that *i)* all the QI-groups in which an individual's records appear have the same set of private values, and *ii)* each QI-group does not contain records having the same private value. However, the application of $m$-invariance to network flow logs is unfeasible, since the cardinality of IP addresses is very large; this would result in a very coarse generalization of QI-values, and in the introduction of a large number of counterfeit flows to enforce property *i)*.

In order to overcome the above problems, in our technique we adopt a many-to-one mapping among IP addresses and encrypted values; this mapping is consistent across the whole set of network flows.

## III. Problem definition and adversary model

In general, the fact that two specific hosts $A$ and $B$ exchanged some message may be considered confidential information. Hence, since an IP address uniquely identifies its host, we assume that confidential information in a network flow is the set of attributes $\{src\_addr, dst\_addr\}$. Indeed, if we could remove those fields from network logs, no confidentiality violation could reasonably be perpetrated. Unfortunately, removal of IP addresses from logs would completely disrupt the utility of the data. When joined with external information, fields in the network flow other than IP addresses may restrict the candidate set of source and/or destination hosts. For instance, as shown in [8], based on network flow data such as packet and Byte counts, it is possible to identify the Web server originating the request. We state that *a network flow is obfuscated if it cannot be associated with high confidence to its source and destination IP addresses*.

### A. Network flow obfuscation

We denote by $L$ an original set of network flows, and by $L^*$ the obfuscated version of $L$ released by the data publisher. The fields of the flows include a confidential multi-value attribute $A^p = \{src\_addr, dst\_addr\}$, and a set of other fields $A^i = \{A_1, A_2, \dots, A_m\}$ that may be used to infer $A^p$. In particular, as explained in Section II, some flow fields may be exploited to identify $A^p$ based on the hosts' characteristics. In order to characterize those fields, we introduce the notion of *fingerprint Quasi Identifier (fp-QI)*.

**Definition** *1 (*Fingerprint Quasi Identifier (fp-QI)*):* A field of a network flow is denoted as a *fingerprint Quasi Identifier (fp-QI)* if its value, possibly combined with external knowledge about the characteristics of the network hosts, can reduce the cardinality of the candidate set for source or destination IP addresses of the flow in $L^*$.

Clearly, which flow fields act as fp-QI strongly depends on the external knowledge available to the adversary. In order to state that two flows are indistinguishable based on the network hosts' fingerprint, we introduce the following notion.

**Definition** *2 (*Fingerprint indistinguishability*):* Two network flows are *fp-indistinguishable* if their fp-QI values are identical.

Given a flow $f$, and fields $A$, $f[A]$ is the *projection* of $f$ onto $A$; for instance, $f[src\_addr, source\_port]$ is the pair $\langle$source IP address, source port$\rangle$ of $f$. Flows are obfuscated by a defense function $D()$ before being released.

### B. Adversary model

At each release of a set $L^*$, the goal of an adversary is to reconstruct, with a certain degree of confidence, the source and destination IP addresses of flows in $L^*$. The considered adversary model is based on the following assumptions:

1) The adversary may observe $L^*$.
2) The obfuscation function $D()$ is publicly known.
3) The adversary may have external information about the characteristics of the target environment, including the fingerprint of network hosts. For example, the adversary may know the topology of the network to be logged, and the set of services offered by its hosts. This knowledge determines which fields act as fp-QI.
4) The adversary may know in advance where and when the flows will be collected, and may inject flows into the network.

Note that we assume a powerful adversary, that may perform both fingerprinting and injection attacks. However, we claim that those assumptions are reasonable. Indeed, some information about the logged network and hosts can be acquired *after* the release of flows; for instance, by scanning the network to locate services. Moreover, in some cases (e.g., if logs are periodically collected and released from a given network), an adversary may also know in advance the network location and the schedules of flows collection, and send bogus messages to target hosts in the network.

## IV. $(k, j)$-OBFUSCATION DEFENSE

In this section we present our defense technique, and we illustrate the achieved confidentiality guarantees.

### A. Defense strategy

As anticipated in Section II, techniques based on a one-to-one mapping of each IP address in an encrypted value *that is consistent across the whole set of flows* are ineffective when an adversary gets to know the mapping among some IP addresses and their encrypted value. On the other hand, mapping the same IP address to different encrypted values in different flows would effectively counteract those attacks, but would result in an excessive loss of information; i.e., it would be tantamount to suppress the IP address fields from released flows. Hence, we propose a novel technique, named $(k, j)$-*obfuscation*, that enforces a many-to-one mapping among IP addresses and pseudo-random group-ID values, which are substituted in obfuscated flows to the real IP addresses. With this solution, each IP address in the released flows is blurred in a set of at least $k$ possible IP addresses.

Note that, with the above solution, an adversary may still be able to identify the real IP address in the group of possible addresses based on the fingerprint of its host. For this reason, our technique includes a defense against fingerprinting attacks. At first, IP addresses are grouped based on the fingerprint of their corresponding host: IP addresses whose hosts have similar fingerprint are grouped together. Then, the fp-QI values of flows are obfuscated, such that, for each obfuscated flow $f^*$ whose source IP $s$ belongs to an IP-group $\alpha$, there exist other $j \leq k$ obfuscated flows, whose source IP belongs to $\alpha$ but is different from $s$, and that are fp-indistinguishable from $f^*$. This way, even if the adversary knows the hosts' fingerprint, as well as the mapping among IP addresses and group-IDs, he cannot associate each flow to less than $j$ different IP addresses.

### B. Formal definition of $(k, j)$-obfuscation

In the following, we define the properties that a $(k, j)$-obfuscation function must guarantee.

**Definition** *3 ($(k, j)$-obfuscation function):* We denote as $D : \mathcal{L} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{L}^*$ a partial function that transforms a set of network flows by substituting each IP address with a group-ID, and by possibly obfuscating the values of the other fields of the flows. Each IP address is mapped to its IP-group by a function *group-ID*; this mapping is consistent across the whole set of flows. We denote as $f^* \in L^*$ the transformation of $f \in L$ obtained by the application of function $D$. We state that $D$ is a $(k, j)$-obfuscation function if, for each set $L$ of network flows, $L^* = D(L, k, j)$ satisfies the following properties:

- *p1:* Each IP-group contains at least $k$ different IP addresses. Formally, for each group-ID $g$ appearing in a flow $f^* \in L^*$, there exists a set $A$ of at least $k$ IP addresses appearing in a flow in $L$ such that, for each $a \in A$, group-ID$(a) = g$.
- *p2:* Each flow $f^*$ is fp-indistinguishable in a set of at least $j$ flows in $L^*$ originated by distinct IP addresses belonging to the same IP-group.

$D(L, k, j)$ is undefined if the above properties cannot be satisfied; i.e., if $L$ involves less that $k$ different IP addresses (it is impossible to enforce *p1*), or if $L$ contains less than $j$ flows (it is impossible to enforce *p2*).

Table I reports a summary of the notation used in the paper.

TABLE I
SUMMARY OF NOTATION USED IN THE PAPER

| | |
|---|---|
| *fp-QI* | fingerprint Quasi Identifier (Definition 1) |
| *src_addr*, *dst_addr* | fields for source and destination IP |
| $f[A]$ | projection of flow $f$ onto field $A$ |
| $L$ (resp. $L^*$) | original (resp. obfuscated) set of network flows |
| $D(L, k, j)$ | $(k, j)$-obfuscation function (Definition 3) |
| $k$ | minimum number of IP addresses in a group |
| $j$ | minimum number of fp-indistinguishable flows |
| $\tau$ | time granularity used in Algorithm 3 |

## C. Confidentiality guarantees

In the following, we present the confidentiality guarantees enforced by our technique, based on different assumptions about the external knowledge available to an adversary.

*1) Defense against knowledge of the IP mapping function:* As explained in Section II, if an adversary discovers a mapping between the real and obfuscated IP address in one flow, and the IP address encryption is consistent across the whole set of flows, he can decrypt the same IP in any other flow in which it appears: such mappings can be easily discovered through injection. As demonstrated by the following theorem, $(k, j)$-obfuscation counteracts this attack, by ensuring that no less than $k$ different IP addresses are mapped to the same IP-group.

*Theorem 1:* Consider a $(k, j)$-obfuscation function $D$, a set $L$ of original network flows, its obfuscated version $L^* = D(L, k, j)$, and an obfuscated flow $f^* \in L^*$. Suppose that the (obfuscated) source and destination IP addresses of $f^*$ are $\alpha$ and $\beta$, respectively. Suppose also that an adversary got to know the function that maps original IP addresses to their group-IDs. Then, based on the knowledge of that function, he can associate the source and destination IP addresses of $f$ to no less than $k(k-1)$ different pairs of possible addresses.

*2) Defense against fingerprinting attacks:* If an adversary knows the fingerprint of network hosts, he can decrease his uncertainty about the source IP of a flow. The following theorem demonstrates that the $(k, j)$-obfuscation technique protects against fingerprinting attacks.

*Theorem 2:* Consider a $(k, j)$-obfuscation function $D$, a set $L$ of original network flows, its obfuscated version $L^* = D(L, k, j)$, and an obfuscated flow $f^* \in L^*$. Suppose that an adversary has accurate information about the hosts' fingerprint. Then, based on this knowledge, he can associate the source IP address of $f$ to no less than $j$ possible addresses.

*3) Defense against combined attacks:* A more powerful threat to consider is when an adversary knows both the IP mapping function, and the hosts' fingerprint. The following theorem demonstrates that $(k, j)$-obfuscation provides strong protection even under this assumption.

*Theorem 3:* Consider a $(k, j)$-obfuscation function $D$, a set $L$ of original network flows, its obfuscated version $L^* = D(L, k, j)$, and an obfuscated flow $f^* \in L^*$. Suppose that the (obfuscated) source and destination IP addresses of $f^*$ are $\alpha$ and $\beta$, respectively. Suppose also that an adversary got to know the function that maps original IP addresses to their group-IDs, and has accurate information about the fingerprint of network hosts. Then, based on this information, he can associate each pair $\langle$*src_addr*, *dst_addr*$\rangle$ to no less than $j(k-1)$ different pairs of possible addresses.

*4) Defense against linking attacks:* In some cases, an adversary may be able to understand that a flow $g^*$ is the response to a flow $f^*$. Different inferences may be used to *link* request and responses; for instance, by observing that a flow from $\alpha$ to $\beta$ is immediately followed by a flow from $\beta$ to $\alpha$. The following theorem demonstrates that $(k, j)$-obfuscation is effective even against this kind of inferences.

*Theorem 4:* Consider a $(k, j)$-obfuscation function $D$, a set $L$ of original network flows, its obfuscated version $L^* = D(L, k, j)$, and two obfuscated flows $f^* \in L^*$ and $g^* \in L^*$. Suppose that the adversary got to know that $g^*$ is the response to $f^*$. Suppose also that he knows the function that maps original IP addresses to their group-IDs, and has accurate information about the fingerprint of network hosts. Then, based on this information, he can associate the source/destination IP addresses of $f$ and $g$ to no less than $j(j-1)$ different pairs of possible addresses.

## V. ENFORCING $(k, j)$-OBFUSCATION

In this section, we present our technique and algorithms to enforce $(k, j)$-obfuscation of network flows.

## A. Overall obfuscation algorithm

Finding the optimal transformation of flows that satisfies $(k, j)$-obfuscation (i.e., the one that minimizes the generalization of fp-QI values, and the suppression of flows) is an NP-hard problem; indeed, it is well-known that even the basic problem of optimal $k$-anonymous generalization is NP-hard [16]. For this reason, we devised an approximate algorithm; its pseudocode is shown in Algorithm 1. At first (line 2), IP-groups are created by executing Algorithm 2 (Section V-B). Then (line 3), the real IPs in network flows are substituted by the identifier of the IP-group they belong to. After initializing

```
Input: $L$: original set of network flows; $fp\text{-}QI$: set of
       fingerprint Quasi Identifiers; $k$: minimum group size; $j$:
       minimum number of fp-indistinguishable flows; $\tau$: time
       granularity for enforcing fp-indistinguishability.
Output: $L^*$: set of obfuscated network flows.
Obfuscate($L$, $fp\text{-}QI$, $k$, $j$, $\tau$) begin
    IP-groups $\mathcal{G}$; IP-group identifiers $\mathcal{GID}$ := GroupCreation($L$, $fp\text{-}QI$, $k$)
    $L$ := SubstituteIPs($L$, $\mathcal{G}$, $\mathcal{GID}$)
    $L^*$ := $\emptyset$
    foreach IP-group $G_\alpha \in \mathcal{G}$ do
        $L_\alpha$ := GetFlows($L$, $G_\alpha$)
        $L^*_\alpha$ := Bucketize($L_\alpha$, $fp\text{-}QI$, $j$, $\tau$)
        $L^*$ := $L^* \cup L^*_\alpha$
    end
    return $L^*$
end
```

**Algorithm 1**: Network flow obfuscation algorithm

```
Input: $L$: original set of network flows; $fp\text{-}QI$: set of
       fingerprint Quasi Identifiers; $k$: minimum group size.
Output: IP-groups: $G_1, \ldots, G_j$; IP-groups identifiers:
        $GID_1, \ldots, GID_j$.
GroupCreation($L$, $fp\text{-}QI$, $k$) begin
    set of IP addresses $A$ := $\{f[src\_addr], f \in L\} \cup \{f[dst\_addr], f \in L\}$
    if $|A| < k$ then return null
    foreach IP address $a \in A$ do
        foreach feature $\in$ fp-QI do
            feature value $v_f$ := GetFeatureValue($L$, $a$, feature)
            fingerprint feature vector $\overrightarrow{a}$ := AddFeature($\overrightarrow{a}$, $v_f$)
        end
        Hilbert index $a_H$ := ComputeHilbertIndex($\overrightarrow{a}$)
    end
    sorted list of IP addresses $\overrightarrow{A}$ := SortOnHilbertIndex($A$)
    group index $j$ := 0
    for $i$ := 1 to $|A|$ do
        if $(i \% k) = 1$ then
            if $(i + k) < |A|$ then
                $j$ := $j + 1$
                IP-group $G_j$ := $\emptyset$
                IP-group identifier $GID_j$ := CSPRNG()
            end
        end
        $G_j$ := $G_j \cup \overrightarrow{A}_i$
    end
    return $G_1, \ldots, G_j$; $GID_1, \ldots, GID_j$
end
```

**Algorithm 2**: Fingerprint-based group creation

the set of obfuscated flows $L^*$ (line 4), for each IP-group, we take the flows generated by the hosts of its IPs, we enforce fp-indistinguishability by executing Algorithm 3 (Section V-C), and we add the obfuscated flows to $L^*$ (lines 5 to 9). Finally (line 10), we return the set of obfuscated flows.

## B. *Fingerprint-based IP-groups creation.*

The goal of our fingerprint-based IP-groups creation method is to enforce property *p1* of $(k, j)$-obfuscation while preserving the quality of obfuscated data. In order to reach this goal, IP-groups are created by grouping together IPs whose hosts have a similar fingerprint (i.e., they originate similar flows), so that fp-indistinguishability can be more easily enforced. The algorithm to group IPs takes as input the original set $L$ of network flows, the set *fp-QI* of fingerprint Quasi Identifiers, and the minimum group size $k$. It returns the IP-groups and their identifiers. The pseudocode of the algorithm is shown in Algorithm 2; its main operations are the following:

1) If less than $k$ IPs appear as source of a network flow in the original set $L$, it is impossible to create an IP-group of size greater than or equal to $k$. Hence, in this case, $(k, j)$-obfuscation cannot be enforced, and the algorithm terminates (line 3 in Algorithm 2).
2) Otherwise, for each IP, we build a *fingerprint vector* (lines 5 to 8), in which each dimension corresponds to a statistics (mean, standard deviation, ...) about the values of an fp-QI field of its flows. This vector represents the fingerprint of the host having that source IP.
3) We map each fingerprint vector in an integer value by exploiting the Hilbert space-filling curves [17] (line 9). A Hilbert space-filling curve is a function that maps a point in a multi-dimensional space into an integer. With this technique, two points that are close in the multi-dimensional space are also close, with high probability, in the one-dimensional space obtained by the Hilbert transformation. In our case, IPs whose hosts have a similar fingerprint are associated to close Hilbert indices.
4) We sort IPs based on their Hilbert index (line 11), and we create groups by partitioning IPs in groups of size $k$ based on that order (lines 12 to 22); if the last group contains less than $k$ IPs, it is merged with the previous one. Each group is identified by a value calculated by a cryptographically secure pseudorandom number generator (CSPRNG) function (line 18). Finally (line 23), we return the IP-groups $G_1, \ldots, G_j$, as well as their identifiers $GID_1, \ldots, GID_j$.

## C. *Enforcing fp-indistinguishability.*

We enforce fp-indistinguishability by bucketizing the values of fp-QI fields. Bucketization consists in substituting the real value of a field with a multiset of possible values. For instance, to make fp-indistinguishable a set of three flows whose number of bytes are 250, 400, and 250, respectively, we substitute the real values in these flows with $\{250, 250, 400\}$.

The bucketization algorithm considers one group of IPs at a time. It takes as input a set $L_\alpha$ of original flows (whose source IPs belong to the same group $\alpha$), the set of fp-QI fields, the minimum number $j$ of fp-indistinguishable flows, and a time granule $\tau$ (for instance, *one minute*). The algorithm considers flows in slots of one time granule at a time, in order to reduce computational and memory costs. This solution is needed when very large sets of flows are considered, as we do in our experimental evaluation. The algorithm returns the set of obfuscated flows $L^*_\alpha$. Its pseudocode is shown in Algorithm 3; the main steps are the following:

1) At first, we initialize two variables $t_{\text{start}}$ and $t_{\text{end}}$ with

```
Input: $L_\alpha$: original set of network flows whose source IP
       belongs to IP-group $\alpha$; fp-QI: set of fp-QI fields; $j$:
       minimum number of fp-indistinguishable flows; $\tau$: time
       granularity.
Output: $L^*_\alpha$: obfuscated flows.
Bucketize($L_\alpha$, fp-QI, $j$, $\tau$) begin
    $t_{\text{start}}$ := lowest timestamp of flows in $L_\alpha$
    $t_{\text{end}}$ := highest timestamp of flows in $L_\alpha$
    repeat
        foreach flow $f \in L_\alpha$ s.t.
        $f[timestamp] \in [t_{start}, t_{start} + \tau)$ do
            fp-QI feature vector $\overrightarrow{f}$ = $f$[fp-QI]
            Hilbert index $f_H$ := ComputeHilbertIndex($\overrightarrow{f}$)
            sorted list of flows $\overrightarrow{F}$ := SortOnHilbertIndex($L_{\alpha,\tau}$)
        end
        group index $i$ := 1
        group of IPs $G_i$ := $\emptyset$
        number of distinct IPs in $G_i$ $n$ := 0
        for $c$ := 1 to $|L_{\alpha,\tau}|$ do
            if ($\nexists f \in G_i$ s.t. $f$[src_addr] = $\overrightarrow{F_c}$[src_addr]) then
                $n := n + 1$
            end
            $G_i := G_i \cup \{\overrightarrow{F_c}\}$
            if ($n = j$) then $i := i + 1$; $G_i := \emptyset$; $n := 0$
        end
        if ($0 < n < j$) then
            if $i > 1$ then $G_{i-1} := G_{i-1} \cup G_i$; $G_i := \emptyset$
            else SuppressFlows($G_i$)
        end
        $L^*_{\alpha,\tau} := \emptyset$
        foreach group of flows $G$ do
            $G^*$ := BucketizeFp-QI-values($G$)
            $L^*_{\alpha,\tau} := L^*_{\alpha,\tau} \cup G^*$
        end
        $t_{\text{start}} := t_{\text{start}} + \tau$
    until $t_{start} > t_{end}$
    return $L^*_{\alpha,\tau}$
end
```

**Algorithm 3**: Bucketization of fp-QI fields

the lowest and highest timestamp of flows in $L^*_\alpha$, respectively (lines 2 and 3 in Algorithm 3).

2) For each flow $f$ in the original set having source IP belonging to group $\alpha$ and timestamp in the interval $[t_{\text{start}}, t_{\text{start}} + \tau)$, we build a *fp-QI feature vector* $\overrightarrow{f}$, in which each dimension corresponds to the value of an fp-QI field of the flow (line 6). We map each fp-QI feature vector in an integer value $f_H$ by exploiting the Hilbert space-filling curves (line 7), and we sort flows based on their Hilbert index (line 8).
3) We partition flows in groups based on their Hilbert order, ensuring that each group contains at least $j$ flows having distinct source IPs (lines 10 to 19). If the diversity of source IPs in the last group is less than $j$, we merge it with the second-last group (line 21). If diversity cannot be enforced, these flows are suppressed (line 22).
4) For each group, we substitute the fp-QI values of its flows with buckets including the fp-QI values of each flow (lines 25 to 28). We repeat the above steps for the subsequent time intervals, until $t_{\text{end}}$ is reached. Finally, we return the set $L^*_\alpha$ of obfuscated flows (line 31).

### D. Correctness and computational complexity

*Theorem 5:* Algorithm 1 correctly computes a $(k, j)$-obfuscation function.

Due to the typically large dimension of network flow datasets, the defense algorithm needs to have low computational complexity. The most computationally expensive task of our algorithm is sorting, which is performed both for grouping IP addresses, and for bucketizing fp-QI fields (complexity in the average case is $O(n \log n)$, where $n$ is the maximum between the number of IP addresses appearing in the flows and the number of flows to be made fp-indistinguishable). The calculation of the Hilbert index is an $O(1)$ operation; it is executed $(n + m)$ times, where $n$ is the number of flows, and $m$ is the number of distinct IP addresses appearing in flows.

## VI. Experimental evaluation

The goal of our experiments was to evaluate the role of $(k, j)$-obfuscation parameters on the utility of obfuscated NetFlows. Of course, higher values of $k$ and $j$ determine stronger confidentiality protection but lower utility of obfuscated flows. In the following, we describe the dataset used in our experiments, and the achieved results.

### A. Experimental setup

In order to carry out our experiments, we collected real traffic packets flowing through an important Italian transit tier II Autonomous System located in the city of Rome, that hosts several sensible corporate and governmental sites. During a day, an average value of about 2 billions of packets, corresponding to about 5 TBytes of data, flow through that system. In order to handle such large amounts of data, it is a common practice to summarize packets into *network flows*. In order to do so, we used the widely adopted CISCO$^{\text{TM}}$ NetFlow[1] technology, which aggregates data obtained from layers $2 - 4$ of the TCP/IP stack. The use of the NetFlow technology has several advantages with respect to raw packet sniffing, since it gives a lightweight and informative picture of the monitored network. NetFlow records proved to be useful for several applications, including intrusion detection systems, traffic classifiers, traffic accounting, DoS monitoring.

The typical configuration to leverage the NetFlow protocol is made of a router with NetFlow capabilities and a probe able to summarize and store received data. A netflow record (graphically illustrated in Fig. 1) is defined as a unidirectional sequence of packets, all sharing source and destination IP address and port, IP protocol, ingress interface, and IP type of service. Other valuable data associated to the flow, like timestamp, duration, number of packets, and number of transmitted bytes, are also recorded; the packet payload is not recorded. In our monitored network, during a typical working day, we collect an average value of about 110 millions of NetFlows.

[1]http://www.cisco.com/web/go/netflow

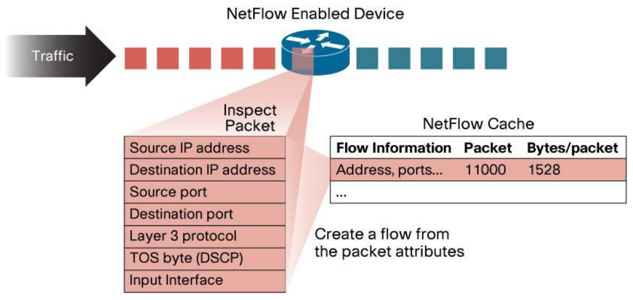


Fig. 1. NetFlow

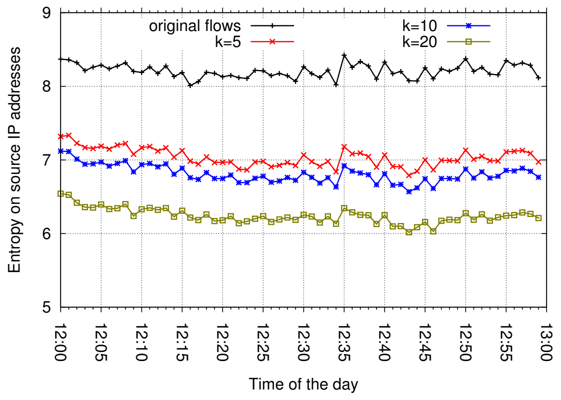


Fig. 2. Entropy of source IP addresses distribution during one hour

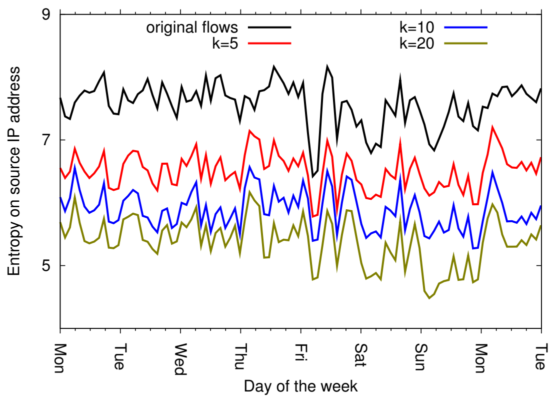


Fig. 3. Entropy of source IP addresses distribution during 8 days

We implemented Algorithms 1, 2, and 3 using *C* and *Python* programming languages. Experiments were carried out on a workstation with IA64 Core i7 930, 2.80 GHz CPU ($4$ cores, $8$ threads), and $12$ GBytes DDR3 1066MHz of RAM, running a GNU/Linux kernel 2.6.32 OS. With this experimental setup, in a few hours we were able to obfuscate NetFlows collected during an entire working day. This is an acceptable time, since, for most applications, NetFlow obfuscation can be performed offline. An extension of our algorithms to support larger sets of NetFlows will be investigated in future work. For the sake of these experiments, we considered a model in which the adversary may have an in-depth knowledge of the network hosts' fingerprint. In particular, we assumed that the fp-QI fields of NetFlows are: type of service (*tos*), protocol, TCP flags, number of packets, and dimension in Bytes.

### B. Impact of parameter $k$: IP address grouping

The first set of experiments was aimed at evaluating the role of parameter $k$ of $(k, j)$-obfuscation; i.e., the minimum dimension of IP address groups. In order to study the effect of $k$ in isolation, we applied Algorithm 2 to the original set of NetFlows in order to partition IP addresses in groups of dimension greater than or equal to $k$. Then, we substituted each real IP address in original NetFlows with its corresponding group-ID. The value of $k$ determines the level of obfuscation of IP addresses. Due to the high complexity of the optimal algorithm for $(k, j)$-obfuscation, and the very large size of the dataset, we could not compare our defense algorithm with the performance of the optimal one. Instead, we have studied the impact of our defense algorithm on obfuscated NetFlows using an information theory perspective; in particular, we measure the network flows entropy Indeed, the network flows entropy evaluation is widespread in traffic analysis, for instance, for traffic anomaly detection [18], and traffic classification [19].

We modeled the distribution of IP addresses in flows collected during one-minute long time windows, in order to evaluate its temporal trend, both in original and in obfuscated flows. High values of entropy are correlated to high diversity of the IP address distribution of flows. Hence, this measure is important for network analysis: for instance, a distributed denial of service attack would determine low entropy on destination IP addresses (many flows are directed to the same host), and high entropy on source IP addresses (many different hosts are performing the attack). Fig. 2 shows the result during a representative peak hour for Internet traffic (from noon to 1 PM), using the dataset of about 7.5 million NetFlows that were collected by our system. We obtained similar results considering other time intervals; in Fig. 2 we plot a one-hour sample for the sake of readability. We executed the IP address grouping algorithm with values of $k$ ranging from $5$ to $20$. As expected, the average value of entropy is inversely correlated to the value of $k$: indeed, the more IP addresses are grouped together, the less diverse the traffic and, consequently, the lower the entropy. However, algorithms for traffic analysis rely on fluctuations of the entropy value; not on its absolute value. For instance, in [18], traffic anomalies are detected by comparing the entropy in a fine-grained time window (e.g., from 12:00 to 12:01 of the current day) with its expected value (e.g., the entropy calculated on the same minute during multiple days). As it can be seen from Fig. 2, trends and temporal patterns are preserved by the transformation of real source IP addresses in group-IDs; we obtained analogous results considering destination IP addresses. These results were confirmed when we considered the distribution of IP addresses in the dataset of about 790 million NetFlows collected during 8 consecutive days; results are illustrated in Fig. 3.

The above results indicate that our technique for IP address grouping preserves both traffic diversity and data utility for algorithms based on information theory measures. For the

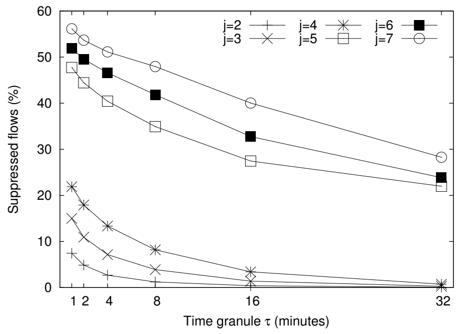


Fig. 4. Suppressed flows ($k = 10$)

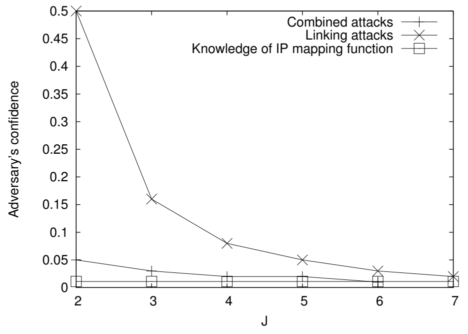


Fig. 5. Adversary's confidence based on different attacks ($k = 10$)

following experiments, we fixed the value of $k$ to 10, since it provides a good tradeoff between confidentiality protection and data utility.

### *C. Impact of parameter $j$: fp-indistinguishability*

The second set of experiments was aimed at evaluating the impact of parameter $j$ on the data quality of obfuscated NetFlows. As explained in Section V, in order to reduce computational and memory costs, our algorithm to enforce fp-indistinguishability takes a temporal granularity $\tau$ as an additional parameter: NetFlows are processed by Algorithm 3 in slots of one time granule at a time. Using shorter time granules demands for less computational and memory resources. However, in some cases, fp-indistinguishability cannot be enforced, since there is no sufficient diversity of IP addresses in flows generated during a single time granule. In those unfortunate cases, our algorithm suppresses those flows that cannot be made fp-indistinguishable.

We performed experiments to evaluate the number of suppressed flows, using different values of $j$ (from 2 to 7) and $\tau$ (from one minute to 32 minutes). With our experimental setup, we were unable to use values of $\tau$ of one hour or more without incurring significant delay, due to swapping, for the specific hardware used in our setup. As it can be seen in Fig. 4, with low values of $j$ (less than 5), the percentage of suppressed flows rapidly decreases; it is close to zero with $\tau$ equal to 32 minutes. On the contrary, with higher values of $j$, a relevant fraction of flows ($> 20\%$) is suppressed. However, with $k = 10$, small values of $j$ are sufficient to provide confidentiality protection. In Fig. 5 we report the adversary's confidence based on the attacks considered in Section IV-C. As it can be observed, with $j = 4$, the confidence of the adversary about the association between a flow and its source and destination IP addresses is below $10\%$.

We evaluated the utility of obfuscated NetFlows in terms of the precision in answering aggregate queries. For those fields having numerical domains (number of bytes and number of packets), we executed the queries considering ranges of different selectivity: e.g., *"count the number of NetFlows at minute $t$ whose number of packets is between* 200 *and* 300*"*. We performed these queries considering each interval of dimension 100 (for the number of bytes) and 5 (for the number of packets), starting from 0, until the maximum dimension of bytes and packets in our dataset of NetFlows. For those fields having non-numerical domains (tos, protocol, and TCP flags), we executed queries about their specific values; e.g., *"count the number of NetFlows at minute $t$ whose protocol is* TCP*"*. We executed queries for each possible value/range, and for each minute in a one-hour time window, for a total of about $120,000$ queries. For each query, we calculated the error rate by the following formula:

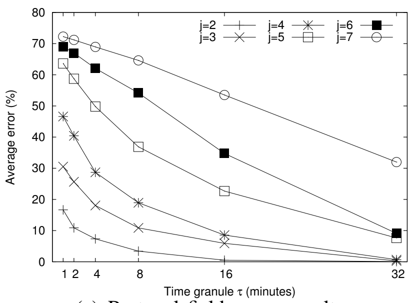
$$e = \frac{\left|\frac{r}{t} - \frac{r'}{t'}\right|}{\frac{r}{t}}$$

where $r$ (resp. $r'$) is the result of the query on the original (resp. obfuscated) flows, and $t$ (resp. $t'$) is the total number of original (resp. obfuscated) flows.

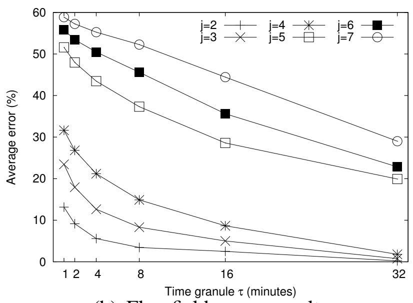
Fig. 6 shows the average error rate for different values of $j$ and $\tau$, and the different fp-QI fields having non-numerical domains. Considering flag and protocol fields, with $j$ equal to 4 or less, and $\tau$ equal to 16 minutes or more, the average error is below $10\%$. The average error rate was even smaller when we considered numerical fields (results are shown in Figure 7). We obtained a larger average error with the tos field; however, even with that field, the error becomes low using $\tau$ = 32 minutes and $j \leq 4$. The average error increases considerably when larger values of $j$ are used; this is due to the large number of flows that must be suppressed to achieve fp-indistinguishability.

## VII. Conclusions and future work

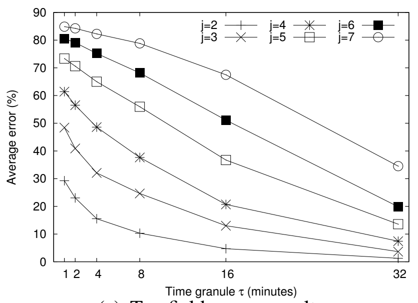
In this paper, we addressed the challenging research issue of sensitive data obfuscation in network flows. We have formally modeled this issue, and proposed a novel defense technique. Differently from previous proposals, our technique provides formal protection guarantees under realistic assumptions about the adversary's knowledge. An extensive experimental evaluation with a large set of real network flows showed that our technique preserves the utility of network flows. Future research work includes the investigation of algorithms for enforcing $(k, j)$-obfuscation on arbitrarily large sets of network flows, and the execution of new experiments on obfuscated flows using state of the art attack-detection algorithms to evaluate data utility. We are also investigating an extension of our defense to different adversary models; in particular, one in which the hosts' fingerprint may change over time.



(a) Protocol field query results

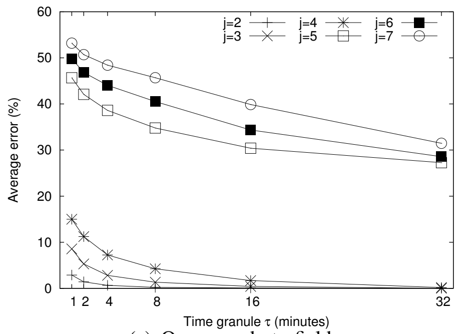


(b) Flag field query results

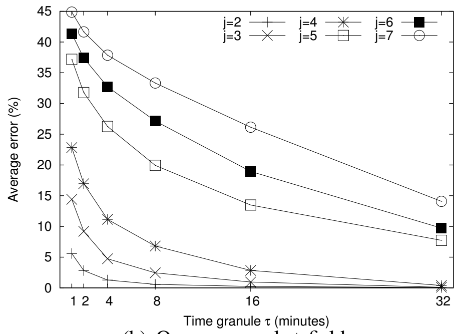


(c) Tos field query results

Fig. 6. Average error rate for aggregate queries on obfuscated NetFlows ($k = 10$)



(a) Query on byte field



(b) Query on packet field

Fig. 7. Average error rate for aggregate queries on obfuscated NetFlows ($k = 10$)

## Acknowledgments

We would like to give a special thank to the CASPUR Consortium and to the staff of its Network Group for the significant help in collecting the data and in the processing of our experiments. This paper has been financially supported by the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme, European Commission -Directorate General Home Affairs, under the ExTraBIRE project, HOME/2009/CIPS/AG/C2-065.

## References

[1] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon, “Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-based scheme,” *Comput. Netw.*, vol. 46, no. 2, pp. 253–272, 2004.

[2] J. King, K. Lakkaraju, and A. J. Slagell, “A taxonomy and adversarial model for attacks against network log anonymization,” in *Proc. of ACM SAC*. ACM, 2009, pp. 1286–1293.

[3] T. Brekne, A. Årnes, and A. Øslebø, “Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies,” in *5th Workshop on Privacy Enhancing Technologies*, vol. 3856. Springer, 2006, pp. 179–196.

[4] R. Pang, M. Allman, V. Paxson, and J. Lee, “The devil and packet trace anonymization,” *Computer Communication Review*, vol. 36, no. 1, pp. 29–38, 2006.

[5] A. J. Slagell, K. Lakkaraju, and K. Luo, “FLAIM: A multi-level anonymization framework for computer and network logs,” in *Proc. of Large Installation System Administration Conference*. USENIX, 2006, pp. 63–77.

[6] M. Foukarakis, D. Antoniades, and M. Polychronakis, “Deep packet anonymization,” in *Proc. of EUROSEC*. ACM, 2009, pp. 16–21.

[7] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Plattner, “The role of network trace anonymization under attack,” *Computer Communication Review*, vol. 40, no. 1, pp. 5–11, 2010.

[8] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter, “Browser fingerprinting from coarse traffic summaries: Techniques and implications,” in *Proc. of Detection of Intrusions and Malware & Vulnerability Assessment*, vol. 5587. Springer, 2009, pp. 157–175.

[9] T. Brekne and A. Årnes, “Circumventing ip-address pseudonymization,” in *Proc. of International Conference on Computer Communications and Networks*. IASTED/ACTA Press, 2005, pp. 43–48.

[10] S. E. Coull, C. V. Wright, F. Monrose, M. P. Collins, and M. K. Reiter, “Playing devil’s advocate: Inferring sensitive information from anonymized network traces,” in *Proc. of NDSS*. The Int. Soc., 2007.

[11] P. Samarati, “Protecting Respondents’ Identities in Microdata Release,” *IEEE Trans. on TDKE*, vol. 13, no. 6, pp. 1010–1027, 2001.

[12] S. E. Coull, F. Monrose, M. K. Reiter, and M. Bailey, “The challenges of effectively anonymizing network data,” in *Proc. of Conference For Homeland Security*. IEEE Comp. Soc., 2009, pp. 230–236.

[13] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, “l-Diversity: Privacy Beyond k-Anonymity,” in *Proc. of International Conference on Data Engineering*. IEEE Comp. Soc., 2006.

[14] N. Li, T. Li, and S. Venkatasubramanian, “t-closeness: Privacy beyond k-anonymity and l-diversity.” in *Proc. of International Conference on Data Engineering*. IEEE Comp. Soc., 2007, pp. 106–115.

[15] X. Xiao and Y. Tao, “$m$-invariance: towards privacy preserving re-publication of dynamic datasets,” in *Proc. of SIGMOD*. ACM, 2007, pp. 689–700.

[16] A. Meyerson and R. Williams, “On the Complexity of Optimal $k$-Anonymity,” in *Proc. of SIGMOD/PODS’04*. ACM Pub., 2004, pp. 223–228.

[17] A. R. Butz, “Alternative algorithm for Hilbert’s space-filling curve,” *IEEE Trans. Comput.*, vol. 20, pp. 424–426, 1971.

[18] Y. Gu, A. McCallum, and D. F. Towsley, “Detecting anomalies in network traffic using maximum entropy estimation,” in *Proc. of ACM SIGCOMM Internet Measurement Conference*. USENIX Association, 2005, pp. 345–350.

[19] J. Yuan, Z. Li, and R. Yuan, “Information entropy based clustering method for unsupervised internet traffic classification,” in *Proc. of IEEE International Conference on Communications*. IEEE Comp. Soc., 2008, pp. 1588–1592.