On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization

Jun Xu Jinliang Fan Mostafa H. Ammar
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{jx,jlfan,ammar}@cc.gatech.edu

Sue B. Moon
Sprint ATL
1 Adrian Court
Burlingame, CA 94010
{sbmoon}@sprintlabs.com

Abstract— **Even though real-world Internet traffic traces are crucial for network research, only a tiny percentage of traffic traces collected are made public. One major reason why traffic trace owners hesitate to make the traces publicly available is the concern that the confidential and private information may be inferred from the trace. In this paper we focus on the problem of anonymizing IP addresses in a trace. More specifically, we are interested in *prefix-preserving anonymization* in which the prefix relationship among IP addresses is preserved in the anonymized trace, making such a trace usable in situations where such prefix relationships are important. The goal of our work is two fold. First, we are interested in analyzing the security properties inherent in prefix-preserving IP address anonymization. Through the analysis of IP traffic traces, we investigate the effect of some types of attacks on the security of the prefix-preserving anonymization process. We also derive results for the optimum manner in which an attack should proceed which provides a bound on the performance of attacks in general. Second, we observe that an existing scheme used for prefix-preserving anonymization, TCPdpriv, has some drawbacks that limit its use in a large-scale, distributed setting. We develop an alternative cryptography-based, prefix-preserving anonymization technique to address these drawbacks while maintaining the same level of anonymity as TCPdpriv.**

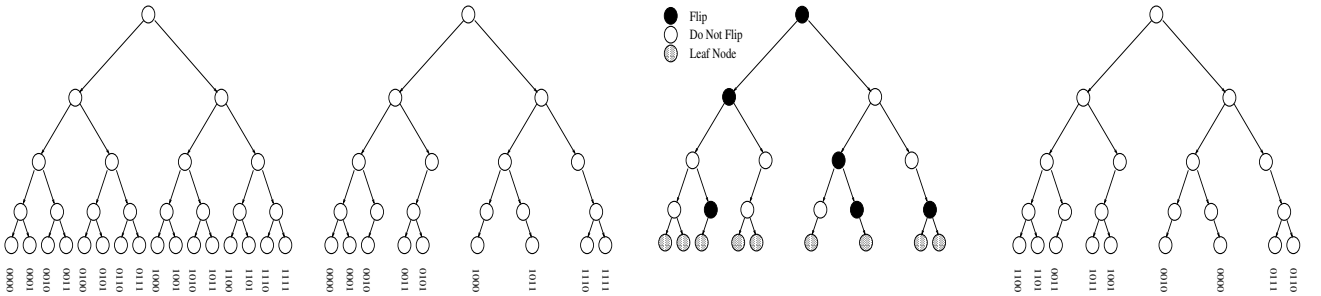
I. Introduction

Real-world Internet traffic traces are crucial for network research such as workload characterization, traffic engineering, packet classification, web performance, and more generally network measurement and simulation. However, only a tiny percentage of traffic traces collected are made public (e.g., by NLANR [1] and ACM ITA project [2]) for research purposes. One major reason why ISPs or other traffic trace owners hesitate to make the traces publicly available is the concern that the confidential (commercial) and private (personal) information regarding the senders and receivers of packets may be inferred from the trace. In cases where a trace has been made publicly available, the trace is typically subjected to an anonymization process before being released.

In this work we focus on the problem of anonymizing IP addresses in a trace. A straightforward approach is one in which each distinct IP address appearing in the trace is mapped to a random 32-bit "address". The only requirement is that this mapping be one-to-one. Anonymity of the IP addresses in the original trace is achieved by not revealing the random one-to-one mapping used in anonymizing a trace. Such anonymization, however, results in the loss of the prefix relationships among the IP addresses and renders the trace unusable in situations where such relationship is important (e.g., understanding routing performance or clustering of end-systems [3]). It is, therefore, highly desirable for the address anonymization to be *prefix preserving*. That is if two original IP addresses share a k bit prefix, their anonymized mappings will also share a k bit prefix. One approach to such prefix preserving anonymization is adopted in *TCPdpriv* [4].

The goal of our work is two fold. First, we are interested in analyzing the security properties inherent in prefix-preserving IP address anonymization. We aim to understand the susceptibility of prefix-preserving anonymization to attacks that may reveal some IP address mappings (e.g., [5]). Through the analysis of IP traffic traces, we investigate the effect of some types of attacks on the security of the prefix-preserving anonymization process. In the process we derive some results pertaining to the optimum manner in which an attack should proceed with the goal of understanding the bounds on the performance of attacks in general. Second, we observe that TCPdpriv has some drawbacks that limit its use in a large-scale, distributed setting. We develop an alternative cryptography-based, prefix-preserving anonymization technique to address these drawbacks while maintaining the same level of anonymity as TCPdpriv.

The rest of the paper is organized as follows. In Section II, we give a somewhat more detailed description of the prefix-preserving anonymization process. In Section



(a) address space (b) original address tree (c) anonymization function (d) anonymized address tree

Fig. 1. Address Trees and Anonymization Function

III we explore how attacks can be modeled and how one may measure the effectiveness of an attack. Section IV provides results from simulation studies of various types of attacks and their effectiveness (as measured by metrics we propose in Section III). For the purposes of this analysis we use publicly available NLANR traces [6] as well as traces collected by Sprint. In Section V we derive a result regarding the optimal form of attack on a prefix-preserving anonymized trace. In Section VI, we discuss TCPdpriv and our cryptography-based technique, which addresses the drawbacks of TCPdpriv. Section VII concludes the paper.

II. Prefix-Preserving Anonymization

As mentioned earlier, with prefix-preserving anonymization, if two original IP addresses share a k bit prefix, their anonymized mappings will also share a k bit prefix. It is useful for our future analysis to consider a geometric interpretation of this form of anonymization. We first note that the entire set of possible distinct IPv4 addresses can be represented by a complete binary tree of height 32. The set of distinct addresses present in an unanonymized trace can be represented by a subtree of this complete binary tree where each address is represented by a leaf. We call this the *original address tree*. Each node in this original address tree (excluding the root node) corresponds to a bit position, indicated by the height of the node, and a bit value, indicated by the direction of the branch from its parent node. Figure 1(a) shows a complete binary tree (using 4-bit addresses for simplicity) and Figure 1(b) shows an original address tree.

A prefix-preserving anonymization function can be viewed as specifying a binary variable for each non-leaf node (including the root node) of the original address tree. This variable specifies whether the anonymization process "flips" this bit or not. Applying the anonymization function results in the rearrangement of the original address tree into an *anonymized address tree*. Figure 1(d) shows the anonymized address tree resulting from using the anonymization function shown in figure 1(c) [1]. Note that an anonymization function will, therefore, consist of I binary variables if the original address tree has I non-leaf nodes.

III. Modeling Attacks and their Effectiveness

Consider now the task faced by an "attacker" presented with a set of IP addresses anonymized with a prefix-preserving algorithm. Because of the prefix preservation property, the attacker can determine the shape of the address tree and hence the value of I. The attacker, therefore, knows *a priori* that he needs to determine I binary variables to completely unravel the anonymization. A complete guess by the attacker without any additional knowledge has a probability of $1/(2^I)$ of being correct.

Typically, however, an attacker operates with the additional knowledge of some mappings of original IP addresses to anonymized addresses. The security implication of prefix-preserving anonymization of traffic traces using TCPdpriv [4] is briefly studied in [5] and [7]. Here we offer a more formal characterization of the security level that a prefix-preserving anonymization can achieve. There are a number of possible ways certain anonymized IP addresses could be compromised in the trace. The port numbers in the TCP/UDP header fields reveals the type of applications at the sending and receiving ends, and provides additional information to attackers. For example, IP addresses of popular web servers can be inferred from their high frequency of occurrence in the trace, and the IP addresses of the DNS servers can be inferred from the hierarchical relationship among them. Suppose certain anonymized addresses have been compromised so that their corresponding raw addresses are known. According

[1] Although what we have presented is clearly a method for prefix-preserving anonymization, it is not immediately obvious that this is the only method. We show in Section VI that this is indeed the only possible anonymization function (Proposition I).

to our geometric interpretation a set of compromised addresses reveals the values for some subset of the I binary variables that specify the anonymization function. Say k of the binary variables are determined by some attacker, the highest level of security we could possibly achieve is to make sure that the remaining $I-k$ binary variables look completely random.

When we study the security of a trace, we would like to measure the amount of information that is leaked from the whole trace as a consequence of compromising some of the address mappings.We define two metrics that characterize this. Suppose we are given a trace file, which contains N distinct anonymized source/destination IP addresses, and each address is 32 bits long. One way to count the number of unknown bits in the trace is to add up all the unknown bits in these addresses, which we refer to as *uncompressed* unknown bits, denoted as U (=$32*N$ when no addresses are compromised). However, due to the prefix-preserving nature of the anonymization algorithm, certain bits in different IP addresses must be equal. If such bits are counted only once, we get another count C (= I when no addresses are compromised), which we refer to as *compressed* unknown bits[2]. According to our geometric interpretation, if we represent distinct IPv4 addresses in a trace as leaf nodes on a binary tree of height 32, then C is count of all their ancestors in the tree. If an address is compromised, then obviously all ancestors (prefixes) of the leaf node that correspond to the compromised address are compromised and should be taken away from C.

IV. Effect of Compromised Addresses on Prefix-Preserving Anonymization

In this section we evaluate the effect of compromised addresses on the U and C measures using two traces:

- An 800MB address trace from NLANR [6] containing 130,163 distinct addresses.
- A 50GB header trace from Sprint containing 1,129,838 distinct addresses. The table below shows the parameters of the trace.

Start Time	09:56 PDT 8/9/2000
End Time	19:56 PDT 8/9/2000
Number of Packets	567,680,718
Location	Packet-Over-SONET OC3 link

Figure 2(a) shows the number of nodes in each level of the original address tree built from NLANR trace. The figure shows that the number of nodes increases when level increases. It also shows that the tree is quite dense on top part but becomes sparser as it progresses towards the leaves representing the IP addresses. Similar figures are obtained from the Sprint trace and are shown in figure 2(b).

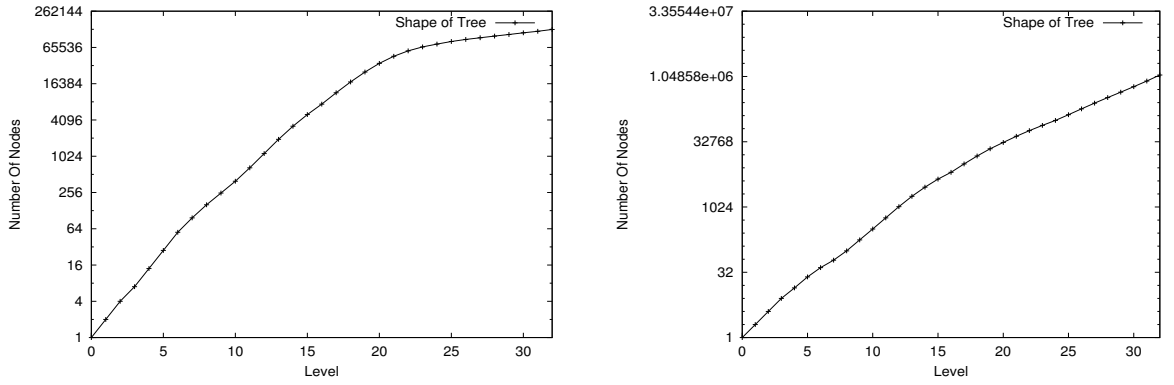
Figures 3(a) and 3(b) are simulation results on the NLANR trace and show how U and C metrics decrease as the number of compromised IP addresses increases. Figure 3(b) magnifies the portion of 3(a) when the number of compromised IP addresses ranges from 0 to 1000. The results are obtained by randomly choosing a certain number of addresses from NLANR trace and evaluating the C and U measures assuming they are compromised. This is repeated 10 times and the graphs represent the average of the experiments.

The value of C drops almost linearly with respect to the number of compromised IP addresses, which is not surprising because it does not reflect how other addresses are affected by the compromised addresses. The value of U drops much faster than C because many other "innocent" addresses have their prefixes revealed which they share with the compromised addresses. The Sprint trace contains many more distinct addresses than the NLANR trace does, the simulation on the Sprint trace, however, exhibits similar trends as shown in figure 4.

Each IP address can be broken down into four groups of bits: bits 1 to 8, 9 to 16, 17 to 24, and 25 to 32. We denote the accounting of U on these four groups as U_1, U_2, U_3, U_4 respectively. As shown in figure 5(a) and 5(b), which corresponds to figure 3(a) and 3(b), U_4, U_3, U_2, and U_1 drop at increasingly faster rate as the number of compromised addresses increases. (Figure 5-b zooms to the beginning part of figure 5(a).) This is not surprising because lower order bits are much more likely to be shared by large number of distinct IP addresses than higher-order bits. Figures 5(a) and 5(b) show that U_3 and U_4 drop very slowly with respect to the number of addresses compromised, which means that the chance for the last 16 bits of an IP address to be revealed due to the prefix-preserving nature of the anonymization scheme is not high. This is indeed good news for privacy-conscious network users: the last 16 bits (typically host ID) in general are much more important for personal privacy than the first 16 bits (typically network ID). For commercial confidentiality, the first 16 bits could be more important, but that could be addressed in some other ways (e.g., releasing the trace after a year). The corresponding results for the Sprint trace are shown in figure 6. The same behavior is observed for this much larger trace.

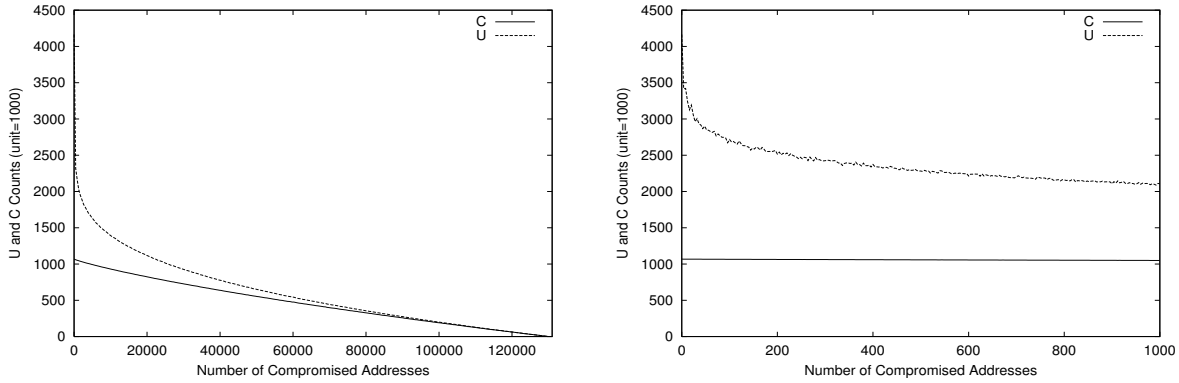
We next consider a similar evaluation for the compressed bits measure, C, which can also be divided into C_1,C_2,C_3 and C_4. The results are shown in figure 7(a) for the NLANR trace and figure 7(b) for the Sprint trace. C_1, C_2,C_3 and C_4 have different initial values and all decrease

[2]This corresponds to the usual sense of entropy



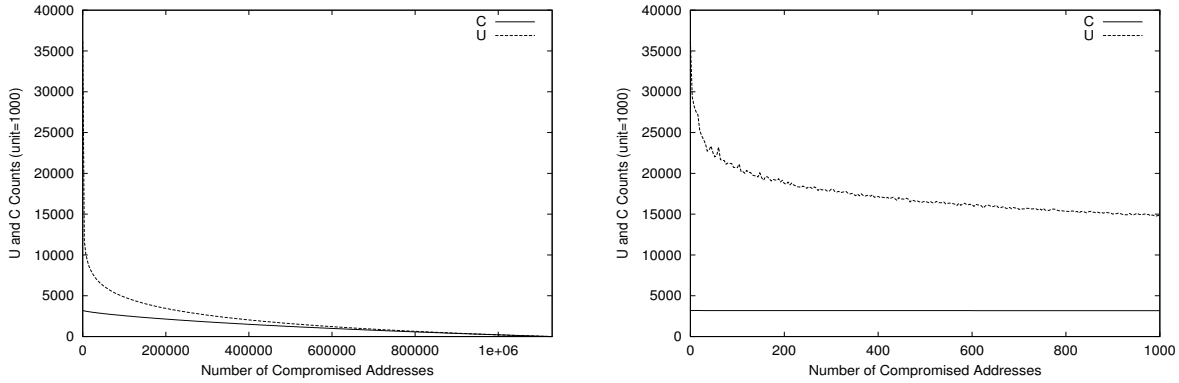
(a) NLANR IP Address Trace (b) Sprint IP Header Trace

Fig. 2. Shape of Address Trees



(a) (b)

Fig. 3. Measurements of U and C on an NLANR IP Address Trace ((b) is magnified version of (a))

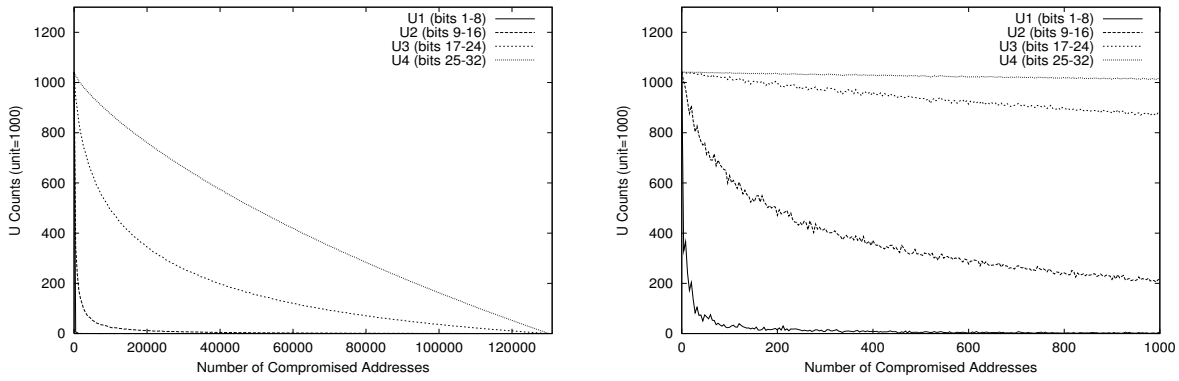


(a) (b)

Fig. 4. Measurements of U and C on a Sprint IP Header Trace ((b) is magnified version of (a))

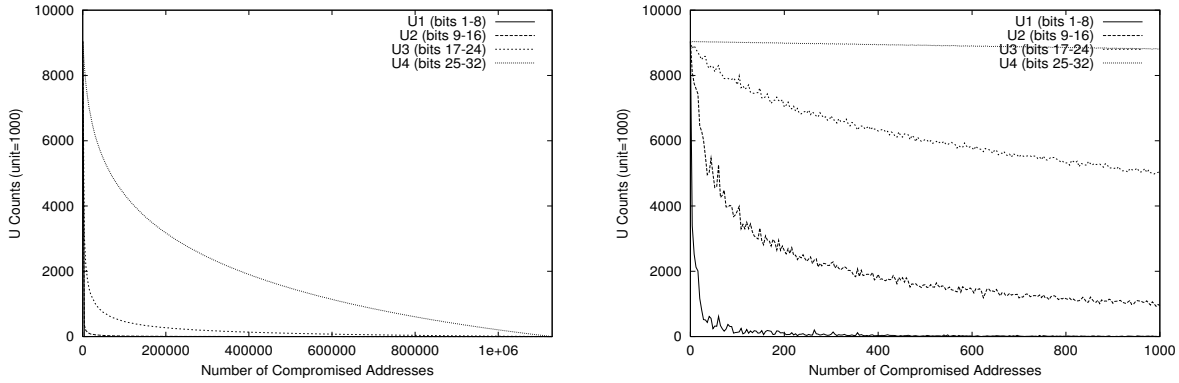
almost linearly as the number of compromised addresses increases.

In the remainder of this section, we examine two examples of attacks on an anonymized trace: frequency analysis and DNS server tracing. Our goal is to understand the extent with which such attacks can compromise addresses. Such an understanding, when combined with our previous analysis of U and C, can yield a better understanding of the security of prefix-preserving anonymization. It should be noted, however, that these attacks apply to not only prefix-preserving anonymized trace but also non-prefix-preserving anonymized traces. Their threat to the prefix-preserving anonymized traces, however, is much more serious than that to the non-prefix-preserving anonymized



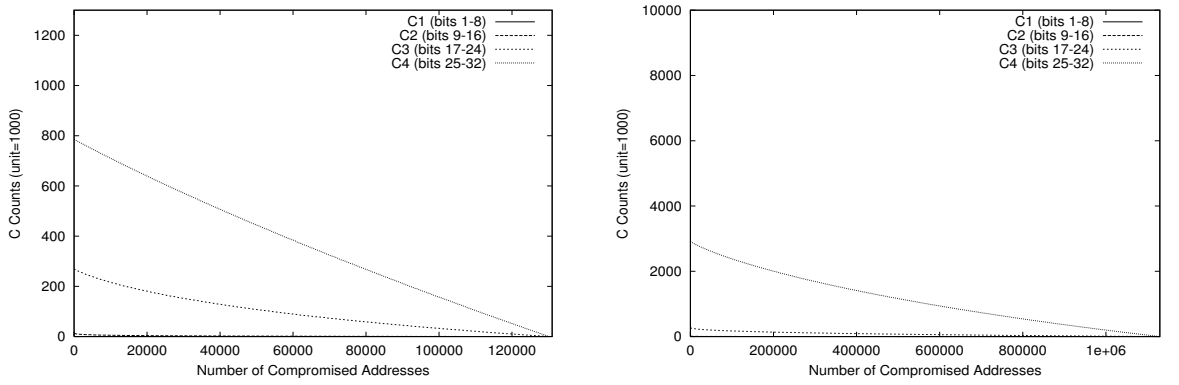
(a) (b)

Fig. 5. Measurements of U(1-4) on an NLANR IP Address Trace ((b) is magnified version of (a))



(a) (b)

Fig. 6. Measurements of U(1-4) on a Sprint IP Header Trace ((b) is magnified version of (a))



(a) NLANR (b) Sprint

Fig. 7. Measurements of C(1-4) on NLANR and Sprint Traces

traces due to the sharing of prefix in the former.

Frequency Analysis: IP addresses of popular sites can be inferred from their high frequency of occurrence in anonymized trace. Figure 8(a) shows the frequency that different addresses occur in the Sprint trace. Addresses are sorted by their frequency of occurrence from left to right. Figures 8(b) and 8(c) magnify the portion of 8(a) when the most frequent addresses range 0 to 1000 and to 100, respectively. These figures shows that only about 25 addresses are actually distinguishable from others by their frequency of occurrence. Referring back to figure 4, it appears that the effectiveness of compromising such a small number of addresses is minor and, therefore, frequency analysis, by itself, does not appear to be a serious threat

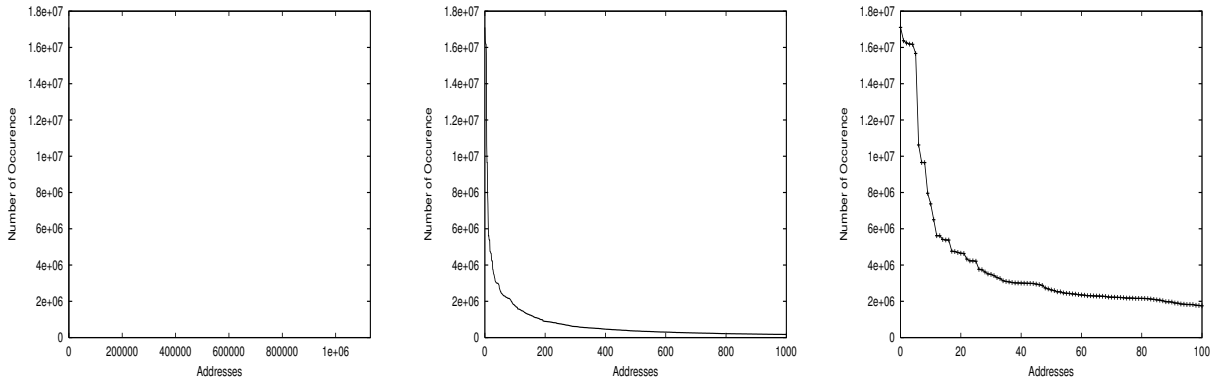


Fig. 8. Addresses' Frequency of Occurrence in Sprint IP Header Trace ((b) and (c) are magnified versions of (a))

to prefix-preserving anonymization.

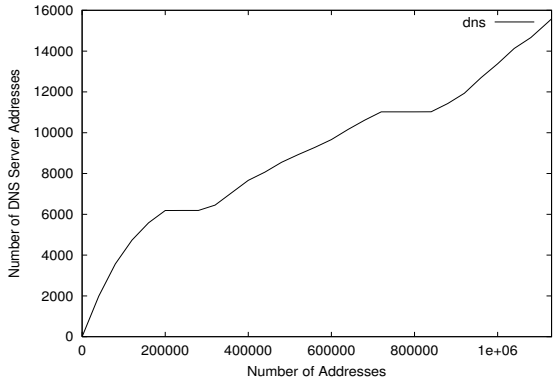


Fig. 9. Number of DNS Server Addresses in Sprint Header Trace

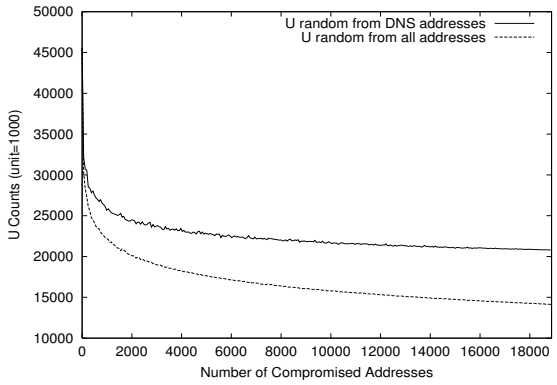


Fig. 10. Effect of Compromised DNS Server Addresses

DNS Server Address Tracing: The IP addresses of the DNS servers can be inferred from the hierarchical relationship among them. Starting with a root DNS server, an attacker can trace down the DNS server hierarchy based on their relationship in the anonymized trace, assuming the attacker has enough knowledge about the DNS server hierarchy.[3]

Figure 9 shows the number of DNS server addresses that appear in a portion of the Sprint trace as a function of the number of distinct addresses, as we consider more and more records in the trace. This figure shows that a proportion in the range of $1/100$ to $1/30$ of distinct addresses in the Sprint trace are DNS server addresses, depending on where the trace is cut. Analysis on some NLANR header traces shows a higher proportion of DNS addresses in the range of $1/10$ to $1/3$. Referring back to figures 4, it can be seen that compromising this many random addresses represents a significant risk to the anonymization process. This might lead one to conclude that an attack that reveals the mapping of all DNS server addresses in the trace will essentially "break" the anonymization process. But this is somewhat misleading since the DNS server addresses are not really random. We investigate this matter further in figure 10 which also derives from the Sprint trace. In the figure we show the value of U as a function of the number of compromised addresses when these compromised addresses are drawn at random from the set of all addresses and when they are drawn at random from the set of DNS server addresses. The figure shows that for the same number of compromised addresses the attacker can reveal more "bits" if the addresses were chosen at random from the entire set of addresses as opposed to the set of DNS server addresses. In fact, compromising all 18,876 DNS server addresses is equivalent to compromising a set of only approximately 1,500 random addresses. This evidence seems to suggest that an attack that reveals DNS server addresses is perhaps not as serious as one would expect and that in

[3]This assumption is quite questionable though. Not all DNS servers allow listing their downstream servers for security reasons. This make it difficult to get the topology of the DNS hierarchy and we have not seen any such topology publicly available yet.

fact an attack that can reveal much fewer random addresses would be more effective.

V. Optimal C and U curves

Every time an addresses is compromised, i.e., a <raw, anonymized> pair is revealed, the values of C and U decrease. In figures 3 and 4 we have shown the decreasing of C and U when N randomly chosen addresses are compromised. These two figures show us the average effect of N compromised addresses. They cannot, however, show us the worst-case effect of k compromised addresses. In this section, we concentrate on discussing *optimal* U curve, which reflects the maximum reduction of the U value when N addresses are compromised. We omit the discussion of the optimal C curve, which basically follows the same way.

A surprising result is that the greedy algorithm (shown later in detail), which generates an address that causes the greatest *single-step* reduction in U value at each step, is actually the optimal solution. That is, N greedily chosen compromised addresses cause the maximum reduction in U curve value among all sets of N compromised addresses. This conclusion is formulated later in Theorem I and proved. The U curves in Figures 11 show the difference between randomly chosen k addresses and greedily (proven optimal) chosen k addresses. As expected, the latter is lower than the former. The differences, however, are minor.

A. Proof of the Optimality of the Greedy Algorithm

In this section, we formally prove the optimality of the greedy algorithm in reducing the U value. This section can be skipped if the reader is convinced of the result. We first introduce the notations and definitions we are going to use in the proof of our main theorem that the greedy algorithm is optimal.

Whenever there is no ambiguity, we denote an address tree (defined in Section II) or a subtree by its root node. We denote the set of leaf nodes on the tree x as $LF(x)$. Given a tree node x, we define $x.left$ and $x.right$ as its left and right child. Also, we denote the cardinality of a set S by $|S|$. Throughout the rest of this paper, n always denotes the number of bits in an IP address (so $n = 32$ in IPv4 and $n = 128$ in IPv6).

Definition I. Given a tree x of height h, we define $R(S, x)$ as the number of address bits, among the last h bits of all leaf nodes (IP addresses), which are revealed when the set S ($S \subseteq LF(x)$) of IP addresses are compromised. Note here that, in computing $R(S, x)$, the first $n - h$ bits are ignored (n is the number of bits in an IP address). $R(S, NULL)$ is 0 by definition. We define $P(S, y, x) = R(S \bigcup \{y\}, x) - R(S, x)$, which is the the number of bits that will be newly compromised when y is added to the compromised address set S.

Definition II. Given a tree x, a set S of leaves is called the *optimal set* defined on x, if given any other leaf set S' of the same cardinality, $R(S', x) \leq R(S, x)$.

Definition III. Given a tree x, a leaf set S is called a *greedy set* defined on x, when the greedy algorithm with input x will generate a sequence of $|S|$ leaves that are exactly elements of S with nonzero probability.

Definition IV. Given a tree x, and a set of leaves S. We refer to $S \bigcap LF(x.left)$ as the left set of S, denoted as $LS(S)$, and $S \bigcap LF(x.right)$ as the right set of S, denoted as $RS(S)$.

With these definitions and notations handy, given a tree x, the greedy algorithm to choose m IP addresses to compromise is formulated in the following. Note that it is a randomized algorithm, since when there are more than one $y's$ that maximizes $P(S, y, x)$, they will be randomly picked with equal probability.

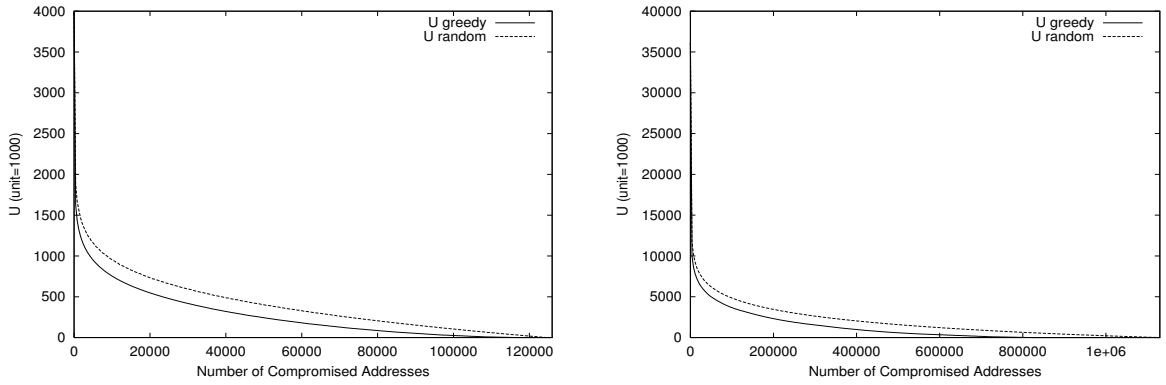
```
Greedy(tree x, int N)
S := ∅
/* S is the set of IP addresses compromised */
R := 0
/* Invariant: R always has the value of R(S, x) */
for i := 1 to N do
    choose y ∈ LF(x) − S that maximizes P(S, y, x)
    /* with randomized tie-breaking */
    R := R + P(S, y, x)
    S := S ⋃ {y}
```

In the following, we prove our main theorem that this algorithm always generates an optimal set. Lemmas used in proving the theorem are introduced and proved afterwards.

Theorem I. Any greedy set is also an optimal set.

Proof: Given a tree x, we only need to prove the following: given any $N > 0$, and any optimal set O and greedy set S of the same cardinality N, $R(S, x) \geq R(O, x)$. This is trivially true when $N = 1$. So in the following, we only consider $N > 1$. We induct on the height h of the tree x. The conclusion trivially holds for $h = 1$ (a single node tree). Suppose the conclusion also holds for $h = k$.

We now prove the theorem for $h = k + 1$ and $N > 1$. Since the elements of set S is a possible sequence generated by the greedy algorithm, they can be ordered according to the greedy order they appear in the sequence. We denote $S_l = l_1, l_2, \cdots, l_i$ as elements of $LS(S)$ in the greedy order, and $S_r = r_1, r_2, \cdots, \cdots, r_{N-i}$ as elements of $RS(S)$ in the greedy order. Similarly, we de-



(a) NLANR (b) Sprint

Fig. 11. Greedy(optimal) vs. Random U Curves

note $LS(O)$ as $O_l = l'_1, l'_2, \cdots, l'_j$ and $RS(O)$ as $O_r = r'_1, r'_2, \cdots, r'_{N-j}$ (no order is assumed in this case). According to Lemma I, the sequences S_l and S_r are greedy sequences in the trees $x.left$ and $x.right$, respectively.

When $i = j$, according to induction hypothesis, $R(O_l, x.left) \leq R(S_l, x.left)$ and $R(O_r, x.right) \leq R(S_r, x.right)$, since the height $x.left$ and $x.right$ are both k. Since $N > 0$, $R(O, x)$ = $R(O_l, x.left)$ + $R(O_r, x.right)$ + $|LF(x)| \leq R(S_l, x.left)$ + $R(S_r, x.right)$ + $|LF(x)|$ = $R(S, x)$ according to Lemma II.

Now we only need to consider the case where $i \neq j$. WLOG, we assume that $i > j$. Then S_r can be extended to a longer greedy sequence $Z = r_1, r_2, \cdots, r_{N-j}$, where r_{N-i+1},$r_{N-i+2}, \cdots, r_{N-j}$ are drawn from $x.right$. Also, $Y = l_1, l_2, \cdots, \cdots, l_j$, being a subsequence of S_l (a greedy sequence in the tree $x.left$ by Lemma I), is also a greedy sequence in the tree $x.left$. Then according to the induction hypothesis, $R(Z, x.right) \geq R(O_r, x.right)$ since $|Z| = |O_r| = N - j$ and $R(Y, x.left) \geq R(O_l, x.left)$ since $|Y| = |O_l| = j$. Define $X = Y \bigcup Z$. Then $R(X, x) \geq R(O, x)$ according to lemma II. Now to prove $R(S, x) \geq R(O, x)$, our final step is to prove $R(S, x) \geq R(X, x)$. We define $Y_m = l_1, l_2, \cdots, l_m$, $Z_m = r_1, r_2, \cdots, r_{N-m}$, and $X_m = Y_m \bigcup Z_m$, where $j \leq m \leq i$. Note that $S = X_i$ and $X = X_j$. What we need to show is $R(X_i, x) \geq R(X_j, x)$.

We claim that $R(X_{m+1}, x) \geq R(X_m, x)$, where $j \leq m \leq i - 1$. We first prove the case where $m = j$. This is equivalent to prove that $P(X_j - \{r_{N-j}\}, l_{j+1}, x) \geq P(X_j - \{r_{N-j}\}, r_{N-j}, x)$. Let G be the set of elements that has already been in the greedy sequence right before the element l_{j+1} is added, when the greedy algorithm is executed to generate S. By the semantics of the greedy algorithm, $G \subseteq Y_j \bigcup Z_i$. So $G \subseteq Y_j \bigcup Z_i \subseteq X_j - \{r_{N-j}\}$. Also $LS(G) = LS(X_j - \{r_{N-j}\}) = Y_j$. Then, since $N > 1$, according to Lemma III(a), $P(G, l_{j+1}, x) = P(X_j - \{r_{N-j}\}, l_{j+1}, x)$, and according to Lemma III(b), $P(G, r_{N-j}, x) \geq P(X_j - \{r_{N-j}\}, r_{N-j}, x)$. Also $P(G, l_{j+1}, x) \geq P(G, r_{N-j}, x)$, since the greedy algorithm chooses l_{j+1} over r_{N-j}. Consequently, $P(X_j - \{r_{N-j}\}, l_{j+1}, x) \geq P(X_j - \{r_{N-j}\}, r_{N-j}, x)$. This proves $R(X_{m+1}, x) \geq R(X_m, x)$ where $m = j$. Using similar arguments, we can prove $R(X_{m+1}, x) \geq R(X_m, x)$ for $m = j + 1, j + 2, \cdots, i - 1$. Then we get our desired result $R(X_i, x) \geq R(X_j, x)$. □

Lemma I. If S is a greedy set defined on the tree x, then $LS(S)$ is a greedy set defined on the tree $x.left$, and $RS(S)$ is a greedy set defined on the tree $x.right$.

Proof: Suppose that $LS(S)$ consists of k nodes $y_1, y_2, \cdots, y_k$ in the sequence it is generated by the greedy algorithm on S. Let G_i be the set of nodes that have been generated by the greedy algorithm right before the node y_i is generated. Obviously $G_{i-1} \subset G_i$. For any i and any $z \in LS(LF(x)) - G_i$, we know that (a) $P(G_i, z, x) \leq P(G_i, y_i, x)$ since otherwise the algorithm running on tree x would not have chosen y_i over z. It remains to show (b) $P(LS(G_i), z, x.left) \leq P(LS(G_i), y_i, x.left)$. That is, the algorithm running on tree $x.left$ (with parameter k) would have a nonzero probability to generate the sequence $y_1, y_2, \cdots, y_k$. We need to discuss two cases. The first case is when $G_i = \emptyset$. In this case, obviously $i = 1$ and y_1 is the first node inserted. Then according to the Lemma IV(a), $P(G_1, z, x.left) = P(G_1, z, x) - |LF(x)|$ and $P(G_1, y_1, x.left) = P(G_1, y_1, x) - |LF(x)|$. Then (b) follows from (a). The second case is when G_i is nonempty (either $i \neq 1$ or G_1 is not empty). In this case, according to Lemma IV(b) $P(LS(G_i), z, x.left) = P(G_i, z, x)$ and $P(LS(G_i), y_i, x.left) = P(G_i, y_i, x)$, (b) also follows from (a). □

Lemma II. Given a tree x, when $|S| > 0$, $R(S, x) = R(LS(S), x.left) + R(RS(S), x.right) + |LF(x)|$.

Proof: Since $|S| > 0$, the height h of the tree x is at least 1. When the first node in S is introduced, the $(n-h+1)_{th}$ bit of every address under tree x is compromised and there are $|LF(x)|$ of them. So $R(S,x) = R(S,x.left) + R.(S,x.right) + |LF(x)|$. However, the last $h-1$ bits of all leaf nodes under $x.left$ and $x.right$ are only affected by $LS(S)$ and $LS(R)$ respectively. That is $R(S,x.left) = R(LS(S),x.left)$ and $R(S,x.right) = R(RS(S),x.right)$. The result follows. □

Lemma III. Given a tree x and two nonempty leaf sets S and T, and a leaf y of x, the following are true:

- (a) If $LS(S) = LS(T)$ and $y \in LS(LF(x))$ (in the left subtree), then $P(S,y,x) = P(T,y,x)$. Similarly if $RS(S) = RS(T)$ and $y \in RS(LF(x))$, then $P(S,y,x) = P(T,y,x)$
- (b) If $S \subseteq T$, then $P(S,y,x) \geq P(T,y,x)$.

Proof: (a) We only prove the first part since the second part follows from the symmetry. Suppose $y \in LS(LF(x))$, then according to Lemma IV(b), $P(S,y,x.left) = P(LS(S),y,x.left)$ $P(T,y,x.left) = P(LS(T),y,x.left)$ since S and T are both nonempty. Then the result follows from the assumption that $LS(S) = LS(T)$. (b) Since $S \subseteq T$, when a new node y is compromised, the set of new bits that are compromised as a consequence in the case when S has been compromised, is a superset of in the case when T has been compromised. The result follows. □

Lemma IV. Given a tree x, if $y \in LS(LF(x))$, then (a) $P(\emptyset,y,x) = P(\emptyset,y,x.left) + |LF(x)|$ and (b) for any nonempty set S, $P(S,y,x) = P(S,y,x.left) = P(LS(S),y,x.left)$. Similarly, if $y \in RS(LF(x))$, then (c) $P(\emptyset,y,x) = P(\emptyset,y,x.right) + |LF(x)|$ and (c) for any nonempty set S, $P(S,y,x) = P(S,y,x.right) = P(RS(S),y,x.left)$.

Proof: We only prove (a) and (b) since (c) and (d) follows from the symmetry. Suppose the height of the tree x is h. First recall that n is the number of bits in an IP address. (a) When y is compromised, the $(n-h+1)_{th}$ bit of every address under tree x is compromised, and there are $|LF(x)|$ of them. Also, for any $z \in RS(LF(x))$, its last $h-1$ bits are not compromised because the length of the common prefix between y and z is no longer than $n-h$. Also, the number of bits (among the last $h-1$ bits) in nodes of $LS(LF(x))$ that will be affected by $y's$ being compromised is fully accounted in $P(\emptyset,y,x.left)$. Therefore, $P(\emptyset,y,x) = P(\emptyset,y,x.left) + |LF(x)|$. (b) This case is similar to (a) except that when S is nonempty, the $(n-h+1)_{th}$ bit of every address under tree x has already been compromised before y is compromised. So the term $|LF(x)|$ does not exist in (b). For the second equality, note that none of the leaf nodes in $RS(S)$ will affect the last $h-1$ bits of the leaf nodes under tree $x.left$. □

VI. Prefix-Preserving Anonymization Schemes

In the following, we describe TCPdpriv, an existing traffic anonymization tool that, among other things, allows the prefix-preservation anonymization of IP addresses. We describe how TCPdpriv implements prefix-preserving anonymization and identify its drawbacks. We then discuss our cryptography-based prefix-preserving anonymization algorithm that does not have these drawbacks. Finally, we precisely define the level of security that is constrained by the prefix-preserving requirement and show that both TCPdpriv and our scheme achieve this level of security.

A. TCPdpriv and Its Drawbacks

TCPdpriv's implementation of the prefix-preserving translation of IP addresses is table based: it stores a set of <raw, anonymized> binding pairs of IP addresses to maintain the consistency of the anonymization. When a new raw IP address a needs to be anonymized, it is first be compared with all the the raw IP addresses inside the stored binding pairs for the longest prefix match. Suppose the binding pair whose raw address has longest prefix match with $a = a_1a_2...a_n$ is <a', b'> (let $a' = a'_1a'_2\cdots a'_n$ and $b' = b'_1\underline{b'_2\cdots b'_n}$), in which $a_1a_2\cdots a_k = a'_1a'_2\cdots a'_k$ and $a_{k+1} = \overline{a'_{k+1}}$. Suppose a is anonymized to $b = b_1b_2\cdots b_n$. Then $b_1b_2\cdots b_kb_{k+1} := b'_1b'_2\cdots b'_k\overline{b_{k+1}}$ and $b_{k+2}b_{k+3}\cdots b_n := RAND(0, 2^{n-k-1}-1)$, where RAND is a pseudorandom (not necessarily cryptographically strong) number generator. If a is not identical to a', a new binding $< a,b >$ will be added to the binding table. It seems that it would take N comparisons to anonymize a new IP address where there are N binding pairs in the table. However, a data structure that is a compressed binary trie in nature is used to reduce the search cost to O(K), where is K is the number of bits (32 in IPv4) in the address. The memory requirement of the algorithm is to store $2*N-1$ trie nodes, where each node occupies 16 bytes. We refer readers to the source code of TCPdpriv [4] for the actual data structure and algorithm.

There are three major drawbacks of this implementation with respect to three different ways traffic traces need to be collected/anonymized for network research:

- First, it is highly desirable to pump out anonymized packet trace in real-time from an operating router, thus saving time and effort for offline anonymization. However, the memory requirement can be high for a long trace with a lot of distinct IP addresses. For example, to anonymize a trace with 10 million different IP addresses in it, it would require approximately 320 MB of main memory space.

Though this is affordable for state-of-the-art computers, such a requirement makes it unsuitable for high-speed hardware implementation inside a router.

• Second, TCPdpriv does not allow distributed processing of different traces simultaneously. The reason is that the translation between the raw and anonymized IP addresses is dependent on the sequence they appear in the trace. So when two different traces are being anonymized, the same raw IP address in general will not be translated into the same anonymized address. However, there is a real need for simultaneous (but consistent!) anonymization of traffic traces in different sites, e.g., for taking a snapshot of the Internet. It would be very cumbersome if hundreds of traces have to first be gathered and then anonymized in sequence.

• Third, a large trace (e.g., terabytes) may be collected for a high-speed link for a long period of time. For the same reason discussed above, TCPdpriv does not allow a large trace file to be chopped into pieces and be processed in parallel. Note that these drawbacks of TCPdpriv remain true even when prefix-preservation is not a requirement.

B. Cryptography-based Approach

We have designed an algorithm that addressed the drawbacks of TCPdpriv by deterministicly maps raw addresses to anonymized addresses. The algorithm is provably secure up to the level of security a prefix-preserving anonymization could possibly deliver. Our algorithm addresses all the drawbacks of the TCPdpriv. Our algorithm "computes" the binding between raw and anonymized addresses on the fly, so that its memory requirement is small enough to be fit into an on-chip cache. This and the fact that is uses hardware-friendly hash functions such as HMAC-MD5 [8] makes it very amenable for hardware implementation. Also, our algorithm is deterministic so that it allows distributed and parallel anonymization of traffic traces. Before we describe our algorithm, we state and prove the following proposition.

Proposition I. Let f_i be a function from $\{0,1\}^i$ to $\{0,1\}$, $i = 0, 2, \cdots, n-1$. Let F be a function from $\{0,1\}^n$ to $\{0,1\}^n$ defined as follows. Given $a = a_1 a_2 \cdots a_n$, then

$$F(a) := b_1 b_2 \cdots b_n \tag{1}$$

where $b_i = a_i \oplus f_{i-1}(a_1, a_2, \cdots, a_{i-1})$ for $i = 1, 2, \cdots, n$ (f_0 is a constant function). We claim that (a) F is a prefix-preserving anonymization function, and (b) a prefix-preserving anonymization function necessarily takes this form.

Proof: It is straightforward to verify the (a) part. We focus on the proof of (b). This is equivalent to proving that given any prefixing preserving F, we can find corresponding $f_i, i = 0, 2, \cdots, n-1$ in the above form. Given any F and any $i, 0 \le i \le n-1$, we define f_i and show that it is well-defined as follows. Given any i-bit sequence $a_1 a_2 \cdots a_i$, we append an arbitrary $n-i$ bit sequence $a_{k+1} a_{k+2} \cdots a_n$ to it. Then we define $f_i(a_1, a_2, \cdots, a_i) := c$, where $c :=$ ($(i+1)_{th}$ bit of $F(a_1 a_2 \cdots a_n)) \oplus a_{i+1}$. Although the definition of c involves $a_{i+1}, a_{i+2}, \cdots, a_n$, from the definition of prefix-preserving, it can be verified that varying the value of $a_{i+1}, a_{i+2}, \cdots, a_n$ in c's definition does not change c's value. Therefore $f_i(a_1, a_2, \cdots, a_i) := c$ is well-defined in the sense that there is only one such c. □

In our algorithm

$$f_i(a_1 a_2 \cdots a_i) := LSB(KH(PAD(a_1 a_2 \cdots a_i), key_i)) \tag{2}$$

where $i = 0, 1, \cdots, n-1$. Here KH is a keyed hash function such as HMAC-MD5 [9], [10] and PAD expands $a_1 a_2 \cdots a_i$ into a 512-bit string as follows:

$$PAD(a_1 a_2 \cdots a_i) := a_1 a_2 \cdots a_i || 10^d || i \tag{3}$$

Here d = 512 - $\lceil log(n) \rceil$ - 1 - i and 0^d means the repetition of 0 for d times. We set aside $\lceil log(n) \rceil$ bits for storing i. So the total length of the data after padding is 512 bits. This padding scheme is standard in using cryptographically strong hash function functions (e.g., MD5 [9]). It guarantees that for two strings a and a', if $a \neq a'$, then $PAD(a) \neq PAD(a')$. WLOG, we also assume that the result of KH is 128 bits as in MD5 (i.e., KH : $\{0,1\}^{512} \rightarrow \{0,1\}^{128}$). It will be clear that the length of the result does not matter.

Later we will show that the proposed scheme is secure even when $key_i, i = 0, \cdots, n-1$ are the same. Using different keys obviously increase the strength of the scheme since it introduces more entropy. However, ironically, to prove the former case, we only need to assume that KH is a pseudorandom function (defined later), which is standard in cryptographic literature . To prove the latter case, on the other hand, we have to assume that KH with different keys satisfy certain independence relationship between them [11]. In the next section, we show that F is provably secure based on the assumption that a secure keyed hash function can be modeled as a pseudorandom function and $key_i, i = 1, 2, 3, \cdots, n$ are the same.

C. Security Analysis of TCPdpriv and Our Scheme

We would like to study the level of security that has been achieved by TCPdpriv and our scheme. We first define precisely the level of security that is achievable by prefix-preserving anonymization algorithms, characterized as follows. Suppose N anonymized addresses have

been compromised so that their corresponding raw addresses are known. Then given any anonymized address b, the first $k+1$ bits of the corresponding raw address is revealed when the longest prefix match between the compromised anonymized addresses and b is k bits long. So the highest level of security we could possibly achieve is to make sure that the remaining $n-k-1$ bits look completely random, i.e., indistinguishable from a $n-k-1$ bit random number.

There are two different levels of security that can be achieved by cryptosystems [12]. One is unconditional security, which means that the cryptosystem can never be broken no matter how much computation power an adversary has. One-time pad is an example of such systems. However, to achieve unconditional security, the encryption key has to be at least as long as the plaintext, which limits the practical use of such cryptosystems. The other level is computational security, in which the security of a cryptosystem is measured by the amount of computation that is needed to break them [13]. A cryptosystem is called computationally secure if the best known method of breaking the system requires an unreasonably large amount of computation time.

For TCPdpriv, if we assume that the sequence in which raw IP addresses get anonymized contains enough natural randomness (they approximately do), or if RAND generates completely random numbers (e.g., using coin tosses), then TCPdpriv achieves exactly the aforementioned level of security unconditionally, even to computationally unbounded adversaries. However, the cost is that it has to remember the big binding pair table, which is essentially an one-time pad. Our scheme, on the other hand, achieves the aforementioned security level when the adversaries are computationally bounded. This will be proved in the next Section.

D. Proof of the Security of Our Scheme

In this section, we prove (later in Theorem II) that our scheme achieves the aforementioned security level when the adversaries are computationally bounded. We will follow the standard notions of security and proof techniques in the provable security literature [14], [15]. Before we state and prove our theorem, we introduce some notations and definitions. For simplicity of discussion, we use U^i to denote uniform distribution on $\{0,1\}^i$. As a convention, random variables and algorithms will be denoted by capital letters and fixed values by lower-case letters. We use "$=_{dist}$" to denote that two random variables are equal in distribution. Again, recall that n always denotes the number of bits in an IP address.

Definition V. (adapted from [16]) Suppose p_0 and p_1 are two probability distributions on the set $\{0,1\}^l$, bit strings of length l. Let $A : \{0,1\}^l \rightarrow \{0,1\}$ be a probalistic (randomized) algorithm. Let $\epsilon > 0$ and two random variables X_0 and X_1 have distributions p_0 and p_1 respectively. We say that A is an ϵ-distinguisher of p_0 and p_1 provided that $|Pr(A(X_0) = 1) - Pr(A(X_1) = 1)| \geq \epsilon$. We say that p_0 and p_1 are ϵ-distinguishable if there exists an ϵ-distinguisher of p_0 and p_1.

Definition VI. We call a function F: $U \rightarrow V$ to be (q,t,ϵ)-pseudorandom, when there is no algorithm A that, given an $x \in U$ at A's choice, can be an ϵ-distinguisher between the uniform distribution on V and the distribution of $F(x)$. Note here that the distribution of $F(x)$ is induced by the distribution of F in function space. So, equivalently, we can say that the function F is ϵ-indistinguishable from a **random function**, which can be viewed as a random variable uniformly distributed in the set of all functions from U to V. Here A is allowed to use F as an oracle on q points of its choice different from x and spends no more than t computation time.

Remark: The following are typically assumed in the cryptography literature [17], [14]. A cryptographically-strong keyed hash function $KH : U \rightarrow V$ can be modeled as a (q,t,ϵ)-pseudorandom function. Here ϵ can be made arbitrarily small, and q and t can be made arbitrarily large by tuning the length of the key (key_i in (2)). For some keyed hash functions such as $HMAC-MD5$, it is believed that this q can be as large as $\frac{|U|}{2}$ without significantly affecting the "randomness" of the "rest" of the function [**?**].

Achieved Level of Security: Now we are ready to define the level of the security that is achieved by our anonymization function F as defined in (1)-(3). Suppose that a set of N anonymized addresses S have been compromised. Given an **arbitrary** anonymized address b (fixed after it is chosen), suppose k is the longest prefix match between b and the elements in S. Then, due the prefix-preserving nature of the anonymization algorithm, the first $k+1$ bits of the corresponding raw address, referred to as a, are revealed as mentioned before. This part is exactly the same as TCPdpriv as explained in Section IV.C. What is different is that the remaining $n-k-1$ bits are indistinguishable from random bits to computationally constrained adversaries (shown in Theorem II); in TCPdpriv, the adversary's computation power can be unbounded. This is actually equivalent to say that the algorithm F is indistinguishable from a **random prefix-preserving function**, a function uniformly chosen from the set of all prefix-preserving functions (characterized in Proposition I) from $\{0,1\}^n$ to $\{0,1\}^n$, to any computationally constrained adversary (to be made precise latter). This means that our algorithm

achieves the highest level of security achievable by prefix-preserving algorithms when the adversary is computationally bounded.

Definition VII Note that S ($|S| = N$) and b can be arbitrary. For simplicity of discussion, in the rest of this section, we consider them fixed once chosen. As we have shown, they in turn determine a and k. Let a be $a_1a_2\cdots a_n$. Given (S,b,a,k), we define $\tilde{F}$: $\{0,1\}^{n-k-1} \rightarrow \{0,1\}^{n-k-1}$ in which $\tilde{F}(x)$ is defined as the last $n-k-1$ bits of $F(a_1a_2\cdots a_{k+1}||x)$, where F is defined as in (1)-(3), and $x \in \{0,1\}^{n-k-1}$.

Theorem II. Given the knowledge of compromised addresses S, if the keyed hash function KH in (2) is a (32*(N+1), t, $\frac{\epsilon}{2^n*n}$)-pseudorandom function, then $\tilde{F}^{-1}$ is a (0,t,ϵ)-pseudorandom function. In other words, given any $y \in \{0,1\}^{n-k-1}$, the distribution of $F^{-1}(y)$ is not ϵ-distinguishable from uniform distribution on $\{0,1\}^{n-k-1}$ for all algorithms A that runs for no more than t time and uses $KH(*, key)$ as an oracle for no more than 32*(N+1) times.

Proof: Since KH is a (32*(N+1), t, $\frac{\epsilon}{2^n*n}$)-pseudorandom function, by Proposition II, $\tilde{F}$ is a (0,t,$\frac{\epsilon}{2^n}$)-pseudorandom function. Then according to Lemma VI, this implies that $\tilde{F}^{-1}$ is a (0,t,ϵ)-pseudorandom function. □

Proposition II. If KH is a (32*(N+1), t, $\frac{\epsilon}{n}$)-pseudorandom function, then $\tilde{F}$ is a (0,t,ϵ)-pseudorandom function even with the knowledge of S.

Proof: We prove the contrapositive. Suppose $\tilde{F}$ is not a (0,t,ϵ)-pseudorandom function. Then there is an algorithm A, which picks an $s \in \{0,1\}^{n-k-1}$ at its choice, can be an ϵ-distinguisher between $\tilde{F}(s)$ and the uniform distribution on $\{0,1\}^{n-k-1}$. Also, A uses no more than t computation time. We need to show if such A exists, then KH is not a (32*(N+1), t, ϵ)-pseudorandom function.

We define $\tilde{F}_i : \{0,1\}^{n-k-1} \rightarrow \{0,1\}$ in which $\tilde{F}_i(x)$ is the i_{th} bit of $\tilde{F}(x)$ for any x, $1 \le i \le n-k-1$. Let $U_i, i = 1,2,\cdots,n-k-1$ be random variables with uniform distributions on $\{0,1\}^i, i = 1,2,\cdots,n-k-1$ respectively. We define random variables $Y_i, 0 \le i \le n-k-1$. For each Y_i, given an outcome ω in the probability space, $Y_i(\omega)$:= $\tilde{F}_1(s)(\omega)$ || $\tilde{F}_2(s)(\omega)$ || $\cdots$ $\tilde{F}_i(s)(\omega)$ || $U_{n-k-1-l}(\omega)$, $i = 1,2,\cdots,n-k-1$. Then Lemma V shows that there exists $l, 1 \le l \le n-k-1$ such that there is a distinguisher algorithm B that satisfies $|Pr(B(Y_{l-1}) = 1) - Pr(B(Y_l) = 1)| \ge \frac{\epsilon}{n-k-1}$. However, as we will show next, this will imply that KH can not be a (32*(N+1), t, ϵ)-pseudorandom function.

Recall that $\tilde{F}(x)$ is defined as the last $n-k-1$ bits of $F(a_1a_2\cdots a_{k+1}||x)$, where $a := a_1a_2\cdots a_n$. We construct an algorithm C that picks $D := PAD(a_1a_2\cdots a_{k+1}s_1s_2\cdots s_l)$ and tries to distinguish the distribution of $KH(D, key)$ from U^{128} (uniform distribution on the range of KH). Given an input $X \in \{0,1\}^{128}$, C first uses $KH(*, key)$ as an oracle $32*N$ times to obtain S (the N pairs of compromised IP addresses) using (2). Then C uses KH as an oracle $l-1$ more times to obtain $\tilde{F}_i(s), i = 1,2,\cdots,l-1$. Then C constructs a random variable Y on $\{0,1\}^{128}$ as follows. Given an outcome ω in the probability space, $Y(\omega)'s$ first $l-1$ bits are $\tilde{F}_1(s)$ $\tilde{F}_2(s)$ $\cdots$ $\tilde{F}_{l-1}(s)$, its l_{th} bit is $LSB(X)$, and its last $n-k-1-l$ bits are $U_{n-k-1-l}(\omega)$. Finally, C returns $B(Y)$ as the result. It is not hard to verify that (a) if X has the distribution $KH(D, key)$ then Y has the distribution of Y_l and (b) if X has the distribution U^{128} then Y has the distribution of Y_{l-1}. Therefore $|Pr(C(KH(D, key)) = 1) - Pr(C(U^{128}) = 1)| = |Pr(B(Y_{l-1}) = 1) - Pr(B(Y_{l-1}) = 1)| \ge \frac{\epsilon}{n-k-1} > \frac{\epsilon}{n}$. This shows that C is an $\frac{\epsilon}{n}$-distinguisher between the distribution of $KH(D, key)$ and U^{128}. Also, C has made no more than $32*(N+1)$ oracle calls and uses no more than t time (evaluting $B(Y)$). This contradicts the assumption that KH is a (32*(N+1), t, $\frac{\epsilon}{n}$)-pseudorandom function. □

In the following we introduce a variation of a standard lemma used in developing the concept of pseudorandom number generator [16]. The original idea behind this proof is attributed to Yao [13]. Let p_0 and p_1 be two probability distributions on $\{0,1\}^m$. Let X_0 and X_1 be two random variables of distribution p_0 and p_1 respectively. We define $m+1$ random variables $Y_i, i = 0,1,\cdots,m$ on the set $\{0,1\}^m$ as follows. Given an outcome ω in probability space, $Y_i(\omega)$:= (the first i bits of $X_0(\omega)$) || (the last $m-i$ bits of $X_1(\omega)$).

Lemma V. If p_0 and p_1 are ϵ-distinguishable, then there exists $l, 1 \le l \le m$ such that the distribution of Y_{j-1} and the distribution of Y_j are $\frac{\epsilon}{m}$-distinguishable.

Proof: It is not hard to verify that $Y_0 =_{dist} X_1$ and $Y_m =_{dist} X_0$. Suppose A is a ϵ-distinguisher between X_0 and X_1. Then $|Pr(A(X_0) = 1) - Pr(A(X_1) = 1)| \ge \epsilon$. However, $Pr(A(X_0) = 1) - Pr(A(X_1) = 1) = \Sigma_{i=1}^{m}(Pr(A(Y_{i-1}) = 1) - Pr(A(Y_i) = 1))$. So according to the triangle inequality, $|Pr(A(X_0) = 1) - Pr(A(X_1) = 1)| \le \Sigma_{i=1}^{m}|(Pr(A(Y_{i-1}) = 1) - Pr(A(Y_i) = 1))|$ (*). Therefore, there must exist $j, 1 \le j \le m$ such that $|Pr(A(Y_{j-1}) = 1) - Pr(A(Y_j) = 1)| \ge \frac{\epsilon}{m}$, since otherwise (*) will not hold. □

Lemma VI. If a permutation $G : V \rightarrow V$ is a (0, t, ϵ)-pseudorandom function, then G^{-1} is a (0, t, $\epsilon|V|$)-pseudorandom function.

Proof: We prove the following contrapositive. Suppose G^{-1} is not a (0, t, $\epsilon|V|$)-pseudorandom function. Let U^V denote the uniform distribution on V. Then there is

an algorithm A, which picks a y_0 at its choice, such that $Pr(A(G^{-1}(y_0)) = 1) - Pr(A(U^V) = 1) \geq \epsilon|V|$. Here A executes no more than t time. We construct an algorithm $B(x, y)$ such that

$$B(x,y) = \begin{cases} A(x) & : \quad y = y_0 \\ 0 & : \quad otherwise \end{cases} \tag{4}$$

Then we construct C such that it can pick an x at its choice and let $Pr(C(G(x)) = 1) - Pr(C(U^V) = 1) \geq \epsilon$. C works as follows. With every execution, C first picks X_0 uniformly randomly from V. Then given any input y, $C(y) := B(X_0, y)$. It remains to show that C is an ϵ-distinguisher. First, we can see that $Pr(C(y) = 1)$=$Pr(B(X_0, y_0) = 1) * Pr(y = y_0)$. So $Pr(C(G(X_0)) = 1)$ = $Pr(B(X_0, G(X_0)) = 1)$ = $\Sigma_{\alpha \in V} Pr(B(\alpha, G(\alpha)) = 1) * Pr(X_0 = \alpha)$ = $\Sigma_{\beta \in V} Pr(B(G^{-1}(\beta), \beta) = 1) * \frac{1}{|V|}$ = $\frac{1}{|V|} Pr(B(G^{-1}(y_0), y_0) = 1)$ = $Pr(A(G^{-1}(y_0)) = 1) * \frac{1}{|V|}$. Also $Pr(C(U^V) = 1)$ = $Pr(B(U^V, G(X_0)) = 1)$ = $\frac{1}{|V|} Pr(B(U^V, y_0) = 1)$ = $\frac{1}{|V|} Pr(A(U^V) = 1)$. Therefore $|Pr(C(G(X_0)) = 1) - Pr(C(U^V))|$ = $\frac{1}{|V|} |Pr(A(G^{-1}(y_0)) = 1) - Pr(A(U^V) = 1)| \geq \frac{1}{|V|} * (|V|\epsilon)$ = ϵ. □

Remark: If the only assumption about G is that it is a pseudorandom function, then this bound of $|V|\epsilon$ for G^{-1} is "almost" tight. To see this, let G be the following (randomized) function. We choose a fixed element $y_0 \in V$ and any subset S of V such that $|S| = \lfloor \frac{|V|}{2} \rfloor$. We also pick a random value x_0 uniformly distributed on S. Then we let $G(x_0) := y_0$, and G restricted on the domain $V - X(\omega)$ be a one-to-one random function from $V - \{x\}$ to $V - \{y\}$. Then it can be shown that G is a (0, ∞, $\frac{1}{|S|}$)-pseudorandom function as $G^{-1}(y_0)$ can be any element in S. However, we will show that G^{-1} is a (0, ∞, $\frac{1}{2}$)-pseudorandom function as follows. An adversary first picks y_0. Then the uniform distribution on S and the uniform distribution on V can be distinguished by the following algorithm A. Given an input x, A outputs 1 if $x \in S$ and 0 otherwise. Let U^V be the uniform distribution on V. Then $Pr(A(G^{-1}(y_0)) = 1)$ = 1 and $Pr(A(U^V))$ = $\frac{|S|}{|V|}$ $\leq \frac{1}{2}$. So $|Pr(A(G^{-1}(y_0)) = 1) - Pr(A(U^V)) \geq \frac{1}{2}$. So when G is a (0, ∞, $\frac{1}{S}$)-pseudorandom function, G^{-1} is not even a (0, ∞, $\frac{1}{2} - \delta$) for any positive δ.

VII. Conclusion

In this paper, we motivate the need for anonymizing packet traces in a prefix-preserving manner and analyze its security implications using real-world traffic data. We show that TCPdpriv, the existing tool that allows for prefix-preserving anonymization, has drawbacks that make it unsuitable for parallel and distributed traffic anonymization. We propose a provably secure cryptography-based scheme that addresses these drawbacks. We expect that this work will help allow more and better Internet traffic traces to be available for network research.

References

[1] Tony McGregor, "NLANR active measurement project," in *Proceedings of Measurement and Analysis Collaborations Workshop*, June 1999.

[2] "The Internet traffic archive," http://ita.ee.lbl.gov/, Apr. 2000.

[3] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *Proc. ACM Sigcomm 2000*, Sept. 2000, pp. 97–110.

[4] *TCPdpriv Command Manual*, 1996.

[5] T. Ylonen, "Thoughts on how to mount an attack on tpcpdriv's "-a50" option ...," in *TCPpdpriv source distribution*, 1996.

[6] NLANR, "File 'sdc-964451101.tstamp+plen+destip' included with NLANR network traffic packet header traces," 2000.

[7] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the wide project," in *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, San Diego, CA, June 2000.

[8] J. Touch, "Performance analysis of MD5," in *sigcomm*, Boston, Massachusetts, 1995, ACM.

[9] R. Rivest, "The MD5 message-digest algorithm," Request for Comments 1321, Internet Engineering Task Force, April 1992.

[10] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, Network Working Group, Feb. 1997.

[11] U. Maurer, "A simplified and generalized treatment of luby-rackoff pseudorandom permutation generators," in *Advances in Cryptology-Eurocrypt'92, LNCS 658*, 1992, pp. 239–255.

[12] A. Menezes, P. Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[13] A. Yao, "Theory and applications of trapdoor functions (extended abstract)," in *Proc. of IEEE FOCS'82*, Nov. 1982, pp. 80–91.

[14] M. Bellare, "Practice-oriented provable-security," in *First International Workshop on Information Security(ISW97)*, Boston, Massachusetts, 1998, Springer-Verlag.

[15] S. Goldwasser and M. Bellare, "Lecture notes on cryptography," .

[16] D. Stinson, *Cryptography, Theory and Practice*, CRC Press, 1995.

[17] Mihir Bellare, "Practice-oriented provable security," in *Lectures on Data Security*, 1998, pp. 1–15.